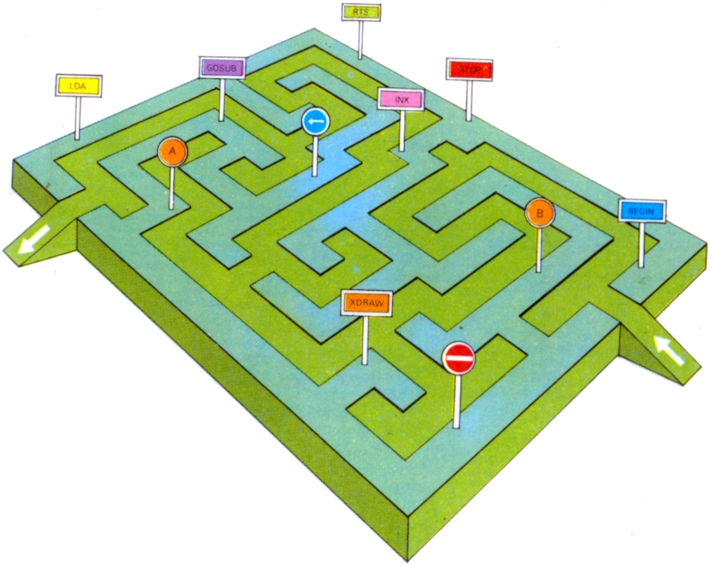
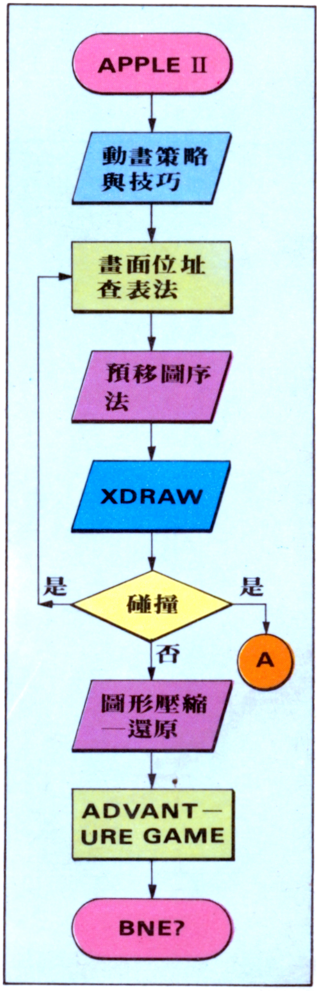
apple II

動畫技巧研究





5 微電腦徹底研究

施威銘 著

暢銷書「第三波」作者托佛勒教授以「第三波」稱呼資訊時代，並以「波前(Wave front)」代表鼓動第三波風潮一些前進的事物。

無疑的，在台灣電腦與管理事業書籍的出版是第三波新文明的一「波前」之一，她將負起引導台灣社會邁向第三波的重大責任。

然而坊間電腦書籍，內容不盡理想，浪費讀者寶貴的時間與金錢。本公司有鑑於此，乃邀請專家學者有系統的撰寫一系列電腦與管理書籍，以最高品質。服務讀者，同時達到「波前出版，必屬好書」的口碑。

波前電腦管理圖書有限公司

封面設計／施威銘

美工完稿／嚴　珣

權所有人	旅娛鈦…… 民國……四月十日 台灣省彰化縣……
發行所及發行人	波前電腦管理圖書有限公司 劉麗雪 民國三十八年三月三十日 台灣省雲林縣
執照字號	台內著字第貳貳伍陸捌號
有效年限	依著作權法第四條規定著作人終身享有著作權並得於其死亡後由繼承人繼續享有三十年但非經註冊不得對抗第三人

部長 林洋港

中華民國七十三年八月 日

內政部著作權執照

著作名稱	著作人		著作權所有人
APPLE II 微電腦徹底研究 第五冊	姓名	施威銘	[illegible]威銘
本著作著作權之保障不包括引用他人之資料。	出生年月日	民國四十二年四月十日	民國四十二年[illegible]
	籍貫	台灣省彰化縣	[illegible]灣省彰化縣

唐凌 序

假如有一天，你用你自己的腦筋，寫了一本書，送到出版社去，或則只是費力地寫好一篇文章，投到某專業性雜誌去發表，你可能遭到一項極大的困難——出版商或雜誌的編輯會向你索取「原文」。

出版商或雜誌編輯之希望文有出處的心理，是不難理解的，一是沾光，一是如有謬誤可一推了之。

事實上，這樣的心態，普通地存在於現居台灣的每一個人心裡，而不只是出版界。

做事有個依據是對的，然而任何依據本身不可能是眞理，即或是眞理，在被引用的過程也可能造成扭曲：

下面就是一段隨處可見而扭曲得可笑的例子：

「在半計數（Counter halves）之間，外來的顫動（Jumper）必須供給。

「若經由Clock 1進入，而且Q 8跳躍至Clock 1，計數比重為1－2－4－5，Q 1是位元比重最大的輸出而且在輸出端有一個對稱的方波。……………

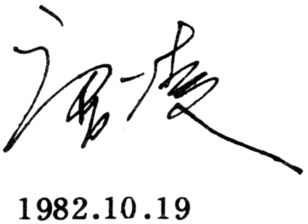
這是一段對 7490 動作的描述，如果你對 7490 毫無概念，那麼看了這一段，眞的會令你「顫動」不已。

不幸的事情是，對這樣荒唐的謬誤，出版商給予容忍，讀者也同樣給予容忍。於是成千上萬的學子們，只好一天天地「顫動」下去。

另外一種情況也是很可悲的，也就是不知從什麼時候開始，大學生都喜歡抱厚厚的洋裝書，使得一些類如三用表用法、OTL、OCL 設計或 CMOS 玩具製作之類的小書，有興趣的人只好偸偸地看，總覺得這些小書，好像很不入流一樣。

我與施威銘先生只是初識，當我看完「APPLE Ⅱ微電腦徹底研究」第一册之後，覺得與我主編的「音響技術月刊」頗有其臭相類之處，那就是作者躍然紙上，隨時與你共處。欲這樣，顯然是需要一些勇氣的，一些面對可能的疏失的勇氣，一些面對抱洋裝書的人的勇氣。我未嘗聽說有人照著食譜做菜，把菜燒焦了、燒爛了、糊了，會找作者理論一番的，但如果你寫技術的書、電腦的書，隨時你都得迎接焦了、爛了、糊了的挑戰。

總之，這不是一篇「序」，而是一點感想，我由衷地希望，每一位讀者都能珍惜作者施威銘先生在這本書中「掏」給你的東西。



1982.10.19

序

有關電腦繪圖的書很多，電腦遊戲的書更多，但是你永遠會覺得，那些書談的與您所見的，都是完全不相同的兩回事！繪圖的問題難嗎？遊戲設計的技巧艱深嗎？爲什麼，我們永遠探究不到問題的核心呢？

精彩有趣的繪圖功能，是個人用電腦的特徵之一，如果我們以傳統電腦的方式來使用，則非但不能發揮其長處，反而是壓抑了它的功能，這也難怪"××電腦繪圖"之類的教本永遠是畫不出小精靈與大嘴巴了。

畫不出小精靈當然不是頂嚴重的事情，可是如果您是生物教師而聽說Apple生動有趣的生物教學，如果您是化學教師而想圖示活化複體的反應過程或是質量作用定律（mass reaction law）的………

在這一本書當中，我們將對 Apple 的電腦繪圖的關鍵技巧予以說明並與大家討論，其內容包含了靜態與動態的彩色繪圖法，而這些方法不獨是使用在遊戲程式裡頭，舉凡可以經電腦畫面顯示的各種應用，這些技法絕對是必須而有效的。早期如小蜜蜂之類的電腦遊戲稱爲Arcade Game（我們俗稱電動玩具），J.stanton 曾出版了"Apple Graphic & Arcade Game Design"一書，有關此類 Game 的基本技巧，本書也將其擇要歸納於第 3 章。但是像 Advanture Game 之類的最新動畫遊戲，其意義又遠超於Arcade Game 了，所以本書在第 6 章當中將這種Game的技法做了相當詳盡的說明。並且，我們也發展出一套很巧妙的基本工具，使得動畫設計不再是一項煩雜無頭緒的工作。

在第 0 章，我們說明了動畫設計的原則，並且以畫面位址查表法來驗證所提出的理論，然後在第 1 章介紹您有趣的彩色技法。第 2 章是繪圖基本動作的詳細解說，尤其強調的是預移圖序的使用，此外還有各種繪圖－

－消圖方式，以及碰撞檢知，第 3 章則擇要地說明了Arcade Game 的構造，並且驗證了各種繪圖——消圖方法，在第 4 章我們介紹一個很有用的繪圖副程式DRAW ，並且熟悉了它的使用法。爲了令程式設計更爲電腦化，在第 5 章您將學到如何令Apple 自動的來繪圖、算圖序表、存圖序表………。當然，在第 6 章，您會學到Advanture Game 有趣的技巧了！

另外，要特別說明的是，爲免佔篇幅，除了少數需要摘要說明的程式外，一切程式說明均載於程式列表的附註上，而書中的主文，則多爲觀念及策略的探討。相信這樣的安排，將能提供讀者更大的視野、更明晰的概念！

總之，這本書，除了獻給所有Apple 的愛好者之外，更希望能對想用電腦繪圖來做其專業工作的教師、工程師、程式設計師………提供一點參考。對寫書的人來說，時間是永遠不夠使用的，如果有著述、校稿、編排上疏失的話，則是因爲我熱切想告訴你的心情吧！

作者　施威銘

目　錄

如何使用本書

第0章　速度的意義

0－0　電腦遊戲——？ ………………………… 3
0－1　速度的意義 ………………………… 5
0－2　不要使用ROM的副程式 ………………………… 7
0－3　畫面位址查表法速度的極限 ………………………… 10
0－4　立竿見影的效果 ………………………… 16
0－5　所以，怎樣來設計動畫程式呢？ ………………………… 20

第1章　彩色繽紛的蘋果

1－0　APPLE Ⅱ 只有六種顏色？——彩色的徹底研究　23
1－1　讓 APPLE 來替我們塗色 ………………………… 26
1－2　將背景移動 ………………………… 43
1－3　垂直與水平的背景移動 ………………………… 50

第2章　故事的主角——預移圖序繪圖法

2－0　畫故事的主角——預移圖序繪圖法 ………… 59
2－1　預移圖序法 ………………………… 62
2－2　彩色圖形的水平移動——彩色的預移圖序 … 65
2－3　畫了圖形之後——各種繪圖法的檢討 ……… 69
2－4　碰撞的問題 ………………………… 77

第3章 ARCADE GAME的構造

3－0 使用畫面位址表：GETADR 副程式 ………… 81

3－1 繪圖程式：XDRAW 副程式 ………………… 83

3－2 讀鍵程式：READKEY ……………………… 86

3－3 搖桿的問題：READPDL …………………… 88

3－4 嘗試寫一段動畫程式 ……………………… 98

3－5 火箭與背景的問題 ………………………… 104

3－6 彩色圖形的水平移動 ……………………… 110

第4章 走吧！大嘴巴！

4－0 預移圖序法的徹底研究 ……………………… 113

4－1 預移圖序的繪圖程序 ……………………… 118

4－2 程式4－0與3－8的比較 ………………… 125

4－3 碰撞及兩種以上圖形的處理法 ………………
走吧！大嘴巴！ ……………………………… 127

4－4 更徹底的來解決問題－－完全的查表法 …… 144

第5章 動畫設計的策略與實現(一) (ARCADE GAME)

5－0 自動的造表程式：SHAPEDIT ……………… 151

5－1 造表程式及其使用法 ……………………… 157

5－2 動畫程式設計的策略 ……………………… 171

5－3　電玩的程式設計　……………………………… 176

第6章　動畫設計的策略與實現(二)
(ADVANTURE GAME)

6－0　什麼是Advanture Game ？　……………… 213
6－1　LUBAR 的構想－－高解像畫面佔8K？　… 214
6－2　HOLLE 的方式　……………………………… 217
6－3　畫面壓縮　－－還原使用例　………………… 223

附錄A　APPLEⅡ高解像畫面結構

A－0　顏色的規則　………………………………… 227
A－1　高解像記憶位址與座標位址的關係　………… 229
A－2　Apple Ⅱ 的畫面解析度是560點？（神秘的 b_7位元）　………………………………… 231

附錄B　偵錯程式

B－0　程式設計的原則　…………………………… 235
B－1　偵錯程式　…………………………………… 236
B－2　LISA 的 TRACE 程式　……………………… 237
B－3　改良的偵錯程式　…………………………… 238

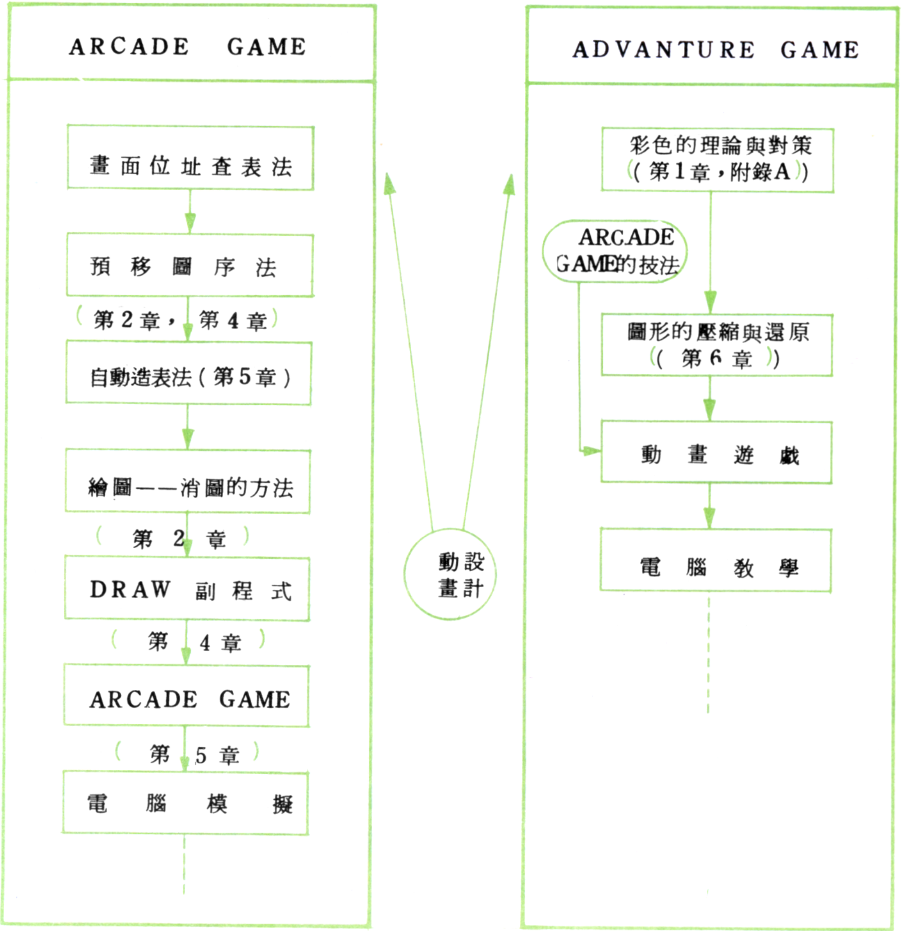
附錄C　電玩的聲音副程式

C－0　視覺與音響共存的秘密　……………………… 241

C－1　改進音響的效果 ………………………………… 243
附錄D　LUBAR的PACK-UNPACK程式 245
附錄E　有趣的英文字母 251
附錄F　計分程式 259
參考文獻 261

為服務讀者，本書所有程式均製成磁片，請向波前公司直接郵購。

本書結構



ARCADE GAME
畫面位址查表法
預移圖序法
(第2章，第4章)
自動造表法(第5章)
繪圖——消圖的方法
(第2章)
DRAW副程式
(第4章)
ARCADE GAME
(第5章)
電腦模擬
動畫設計
ADVANTURE GAME
彩色的理論與對策
(第1章，附錄A)
ARCADE GAME的技法
圖形的壓縮與還原
(第6章)
動畫遊戲
電腦教學

如何使用這本書

本書的目的是希望經由實際程式的觀察、分析，進而整合、改良，最終期能對於動畫設計這種看似龐大困難的問題，有一整體的瞭解與掌握。因此，除了閱讀之外，當然也要實際上將程式鍵入到機器內實作一翻。爲了方便各種不同的使用場合，本書的程式使用法如下：

本書的程式分爲兩種，一種是可以執行的程式，我們在每則程式之後均附有機械碼，另外一種是用以說明程式的架構，當然不能執行，其後也沒有機械碼。

如果：

①您只有Apple II的主機，那麼只要CALL －151參照使用手册的說明，將每則程式的機械碼鍵入即可。除錯的時候，除了將每個16進位碼一一對照之外，也可以利用Apple II監督程式的〞L〞命令（Disassembly）來幫助除錯，如果發現“？？？”便是該位址前後（不一定是該位址本身），發生錯誤了。

②您有逐步執行（One－step）及Trace 的監督程式ROM ，則可用以幫助除錯工作，對於書本所列的程式或許功用不大，但是如果自己發展程式，就是相當必要的工具了，至於自動啓動（Auto-start）ROM的使用者，也可以參考叢書第四册的方法來輔助偵錯。

③您有磁碟機，並且使用了組譯器（ Assembler ），那麼可以直接鍵入本書的組合語言程式，如此對於程式修改、擴展更爲方便。本書的程式是以LISA寫成的，如果您使用了其它的組譯器，則可以依照其使用手册上的說明稍予修改便可以了。下表是較常爲人使用的 DOS TOOL KIT 與 LISA之間的指令對照表：

LISA	Apple Tool Kit
ADR	DW
END	(不必用)
EQU（非零頁定址法）	EQU
EPZ（使用零頁定址法）	EQU
HEX	DFB
ICL	CHN
LST	LSTON
NLS	LSTOFF
PAG	PAGE

LISA 與 Apple Tool Kit 指令互換表（僅列出對應不同者）

LISA	SWEET-16
LDR	LD
STO	ST
PPD	POPD
BRA	BR
BIC	BC
BIP	BP
BIM	BM
BIZ	BZ
BNM	BNM1
BKS	BK
RSB	RS
BSB	BS

LISA 與 WOZNIAK 的 SWEET-16 指令互換表（僅列出不同者）

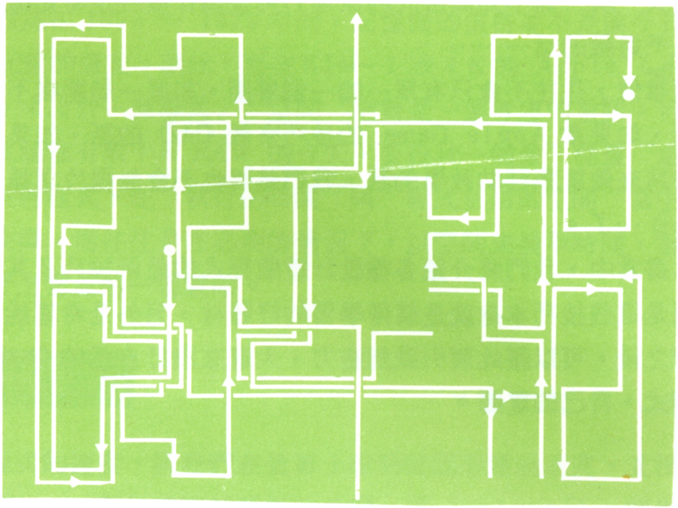
本章提要：

相信是好奇心使您對這本書感到興趣，隨著本書的每一章節，您將會看到或許是您曾想過而不得其解的或是未曾想過新鮮的事，所以請以高昂的興緻來開始我們的第 0 章……………

本章將介紹您：

1. 近代動畫（包括 Game ）設計的基本形態
2. 時間與空間的抉擇
3. 動畫設計的特殊考慮因素
4. 畫面位址查表法
5. 其它

0 速度的意義



0-0 電腦遊戲 ?

如果您在舊書攤翻到一本十年前的電腦書籍，或許可以看到其中有一頁是“ 蒙那麗莎的微笑 ”這樣的畫，這種畫不是用圖點（ Pixel ）畫出來的，通常是用„ ** ”號或其它字母組合而成，如果您還發現到書中有所謂電腦遊戲的話，則不外乎心算、猜字或猜顏色之類的遊戲。

今天，如果有人要你鍵入一個用 5 H 代表紅桃 5 ，8 S 代黑桃 8 的 Game 在 Apple 上玩玩，你一定認爲此君未免不經世面。在一個可以很生動的畫出各樣圖形的 Apple II 上使用 5 H，8 S，就像是當您學會彩色畫圖之後，再也不會用“ RED 代表紅色，BLUE 說是藍色…… ”一樣的道理了。問題是如何才可以設計出優秀的遊戲程式呢？儘管今天，小精靈、小蜜蜂、金撲克的程式隨處可以買到，我們却仍舊迷惑不知其解（或者是已知其“ 解 ”而不知其設計的原理)，可是爲什麼一定要知道其原理呢？如果只是遊戲的話………

任何事情，往往不會只有單一的一個層面，如果一局圍棋不把它想成圍棋的話，如果拿 Apple Game 的技巧來做 CAD 的話，如果將遊戲程式設計者請來做電腦輔助教學的話，我們將再也不會無視於這種大雅之堂以外的………了。

在本書當中，我們將介紹各種設計遊戲程式的最新技巧及其基本原理，一方面是這些技巧本身就是值得學習的好題材，另外是希望能夠經由這樣的學習之後，可以藉此而引發創造力，大家來設計自己的 Game 、自己的電腦程式、自己的電腦。

0-1 速度的意義

時間與空間　新的設計哲學

最初的一些微電腦程式，通常是以如何節省記憶空間爲重要的考慮因素，這是因爲在 1975 ～ 1979 年間，記憶體的價格並不是像今日 16 K RAM值50元般的廉宜。因此，要在KIM－ 1 或 4 K 的 Apple （系統已佔用了不少）來設計一個程式，“空間”絕對是一個考慮的重點。可是在幾乎每部微電腦都有數十K RAM 的今天，要設計一個像“動畫”這樣以速度表現見長的程式，在“執行的時間”與“程式佔用的空間”當中，卻必須有一個抉擇，而往往是在必要的情況下，以空間來換取時間，這是今天 Game 設計上常見的一種策略。

速度爲什麼重要

動畫程式的速度，與我們在螢幕上看到的圖形運動速度，並沒有直接的關係。譬如，一顆快速穿越螢幕的子彈，用 Applesoft 一樣也能達成。反之，一面微風吹動而緩緩飄揚的國旗，却必須要以絕對快速的機械語言才足以表現其悠揚有緻的自然韻味。因此，速度在現代動畫程式設計裡的意義，是在於如何發揚電腦快速的能力來完成畫面動作的每一細節，而不只是單純的圖形運動而已。

舉例來說，要讓畫面上的一個點做直線的移動，我們可以一次向旁邊移動一個圖點（pixel）距離，如此這個點是眞正的經由路徑上的每一位置來移動的，另外，我們也可以間隔 3 個或 5 個圖點來移動，這樣的點是以跳躍的方式來行進的。對於相同的距離而言，前者勢必要比後者花更多的時間來完成全部的動作，下列的程式可以說明我們的看法：

```
]LIST

10  HGR
20 J = 2: GOSUB 100
30 J = 15: GOSUB 100
40  GOTO 20
100  FOR I = 0 TO 277 STEP J
110  HCOLOR= 3
120  HPLOT I,100: HPLOT I + 2,100

125  HPLOT I,101: HPLOT I + 2,101

130  FOR R = 0 TO 7: NEXT R
140  HCOLOR= 0
150  HPLOT I,100: HPLOT I + 2,100

155  HPLOT I,101: HPLOT I + 2,101

160  NEXT I
200  RETURN
```

程式 0 - 0

這個程式在螢幕上畫出由 4 個白色點所組成的方塊，然後分別由第 2 0 行及第30行所設定的J 值來控制此小方塊的水平移動。我們可以發現到，當畫點的間隔加大時（ J ＝ 15），則圖形的移動速度變快，但卻有不連續的現像，而當畫點的間隔變小時，圖形的移動便很自然了。或許您會疑問到，雖然圖形移動自然了，但是速度不也變慢了嗎？是的，這也便是本書的主題所在：“ 如何以快速的畫圖速度（注意不是運動速度），來造出自然而生動的遊戲畫面 ”！

0-2 不要使用ROM的副程式

如何加快程式執行的速度

程式 0 − 0 因爲是使用BASIC 寫成的，速度自然緩慢了，但是一想到使用機械碼這種看似毫無頭緒的語言，便有不知從何著手的感覺，到底設計這樣的程式，是否有些什麼特別的技巧呢？在此，我們將有關於動畫程式設計的一些技巧，以及專業設計上常予考慮的幾個問題，簡述如下，這些問題，我們將分別在往後各章節當中以實例來加以說明，此處特別先予提出，是爲了加強印象以便在以後的實例當中仔細研究。

㈠不使用 ROM 內部的副程式

或許您會覺得奇怪，或不予贊同，以往，我們不是多方介紹 Apple 監督程式或 Applesoft ROM 的副程式，並且加以運用嗎？Apple ROM 裡頭的許多副程式是極其方便的，使用 ROM 副程式可以使我們的程式縮短許多，節省空間也節省程式設計的人力及時間。但是，這種每個人都可使用，亦即適合各種用途的公用程式，在本身的結構上必然會因其多用途的考慮，而週詳的予以設計，這些週詳的設計，便是減慢執行速度的來源。

更直接的來看這個問題，那便是一旦您使用一個 JSR ROM副程式的指令，其實您已經到 ROM 裡頭繞上一大圈了，而這一大圈有許多是爲了程式多用性而加入的指令，可是我們所要的是只執行一件工作的程式而已，這樣的程式，自己來寫自然是專對目標而快速有效了。

所以，不要用 ROM 副程式！一個快速有效的程式，便是“只做目的份內工作”的程式，所以絕不多用一個不必要的指令。多數的 ROM 副程式，並不符合這種要求！（時間延遲程式除外）

㈡不用計算，多用查表

我們知道 Apple 畫面構造較為特別，它並不是依照順序來安排其畫面與記憶位址的對應關係。Apple 記憶位址與畫面座標的關係是如圖 0－1 的公式所示，也就是說每次要在（X，Y）座標上畫一個點，便要經由這組公式計算出相對應於（X，Y）座標的記憶位址的值，然後將圖案資料存入其內，如此才能在螢幕上畫出圖形來。要做上述的計算至少得使用十幾個指令才足以完成，如果是要在 Apple 192 條水平掃描線上塡入圖形，光是位址計算便要花上 2000 多個指令執行時間，對於連一微秒（μs）都必爭的動畫程式，這是相當大的時間浪費。

如果我們事先將上述公式的計算結果，算好列表並存入程式當中，則每次我們只要查出表格中的值，便等於是算出上述的公式，雖然多用了記憶空間却節省了不必要的計算時間。所以，多用查表，不用計算！

㈢不必用完全部螢幕

Apple 的水平位置有 280 點，但是一個記憶位址只能夠存放 0 至 255 的值，因此欲使用全部的 280 個水平座標，必須使用兩個記憶位址來存放其座標值。顯而易見的，如果放棄畫面右邊的24點×192線的畫面空間的話，只要使用一個位址便足夠來存放 X 座標值了。如此至少可以令資料存取的速度加快許多（因為只對一個記憶位址存取資料，並且沒有進位）。這便是某些遊戲畫面，譬如 Apple Invader 或 Apple Galaxy，不是用畫面右邊當計分表，便是畫上一些不動的背景圖案的原因了。

㈣不要用PHA與PLA，多用 0 頁

通常，我們慣用堆疊器（ Stack ）來存放一些暫時的資料，雖然 PHA或 PLA表面上看來是一個位元組的指令，看似簡短快速，其實PHA 及 PLA共要花掉 7 個時脈週期，尤其是要將X，或Y 的值存入堆疊器，則更要使用TXA，TYA 等多重手續，總合起來便要11個時脈週期。如果我們用 0 頁的STA，LDA（或LDY，STX 等）的話，則前後才不過 6 個時脈週

期，便可達成暫存資料的目的了。可是爲什麼有許多程式是使用PHA與PLA呢？這是因爲 Apple 系統本身已幾乎使用了全部的0 頁位址了，因此只好以PHA及PLA來存放暫時資料。不過請別忘記！您現在是設計一個連ROM副程式都不使用的動畫程式，當然許多Apple 系統佔用的0 頁位址，現在均可自由使用了（因爲已不使用 Apple 原來的系統）。所以，不要用PHA與PLA，用0 頁位址！

(五)常出現的事件，不要寫成副程式

雖然在一些基本課程裡，我們知道副程式有許多好處，譬如節省程式空間，使程式明白易讀，減少錯誤發生的機會，使程式結構化……，不過副程式的一個缺點便是浪費時間，尤其嚴重的是多重廻圈——多重副程式的使用，所以程式設計的原則是“常出現的事件，不要寫成副程式”。當然，任何程式當中是免不了要使用副程式的，通常在動畫程式裡頭，我們以在主廻圈裡頭使用副程式爲原則，而盡量避免在副程式中再呼叫其它副程式。另外，我們也很少看到公用一個副程式的場合，否則，這個副程式與ROM的副程式也就沒多大差別了。

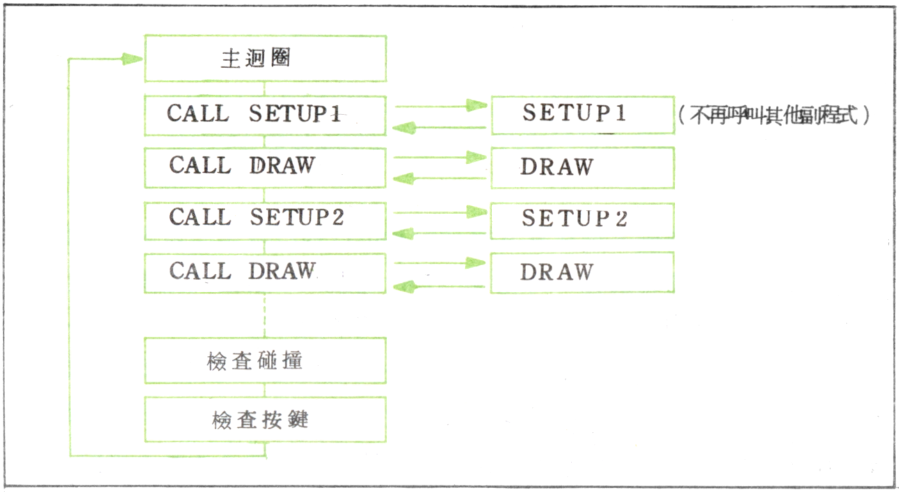


圖 0 - 0

0-3 畫面位址查表法　速度的極限

前一節我們說不要使用ROM的副程式，要寫“ 唯一目的 ”的程式，不要計算螢幕位址，要用查表法。現在就讓我們馬上來證實這種說法。

記得在 Apple ROM 裡頭有一個可以變換畫面底色的副程式 $F3F6 (62454)，您可以由下列程式來加以觀察：

```
]LIST

10  HGR
20  HCOLOR= 3
30  HPLOT 0,0
40  CALL 62454
50  FOR I = 0 TO 500: NEXT I
60  HCOLOR= 2
70  HPLOT 0,0
80  CALL 62454
90  FOR J = 0 TO 500: NEXT J
100  GOTO 20
```

程式 0－1 用ROM副程式來變換畫面底色

執行這個程式的時候，請注意當第40行及80行執行時畫面變化的速度，您仍然可以看出其變化過程，可見 $F3F6 這個ROM副程式速度並不頂快。

那麼怎樣才可以快速的變換畫面底色呢？使用查表法（ Look － up table ）便可以了。依照我們的原則，任何螢幕動作莫快於對記憶位址的直接存取，而欲對螢幕位址做資料存取最快者莫過於查表法了。可是我們需要怎樣的表格呢？全部高解像畫面 8 K 的位址表嗎？其實，我們之所以花時間在這些螢幕工作，Apple 畫面的不規則性（較複雜的規則）是一

大因素，幸運的是，這種不規則性只發生在螢幕垂直座標方向，而水平方向却是連續排列的。因此，只要找到每條水平線的起始位址（左邊端點所對應的位址），那麼右邊接著的所有點的位址便只是依水平位置增加而增加：

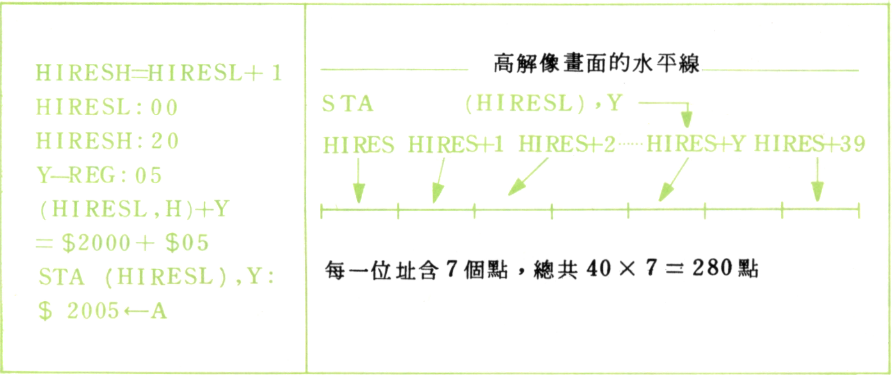


圖0－1 零頁間接索引定址使用法

圖0－1便是說明如何對水平方向位址的存取，只要我們知道最左邊端點的位址 HIRES=(HIRESL,HIRESH)，所有此線上的位址，均可由間接索引定址法 STA (HIRESL),Y 或 LDA (HIRESL),Y 等來加以存取。問題是位址 HIRES 並不依照垂直座標順序排列，HIRES 的值是依下列公式來計算的：

```
Y1=INT(Y/8), Y2=INT(Y1/8)
YR=Y-Y1*8, YS=Y1-Y2*8
HIRES=8192+Y2*40+YS*128+YR*1024
```

圖0－2 高解像位址與畫面座標的關係

每次我們要在畫面上（X，Y）座標畫出一條線段（此處X是以位元組為單位，即水平方向一個位元組所含的7個點）便要通過圖0－2公式的計算，然後將＄FF或＄7F存入到HIRES+X的位址內*。許多高解像畫圖程式便是因為這些計算而影響到執行速度。但是Y座標的值只不過是由0到191共192個而已，如果我們事先將所有的Y值代入圖0－2的公式中，將結果求出並列成表格的話，不就可以節省許多計算時間嗎？因為表格存取，等於是位址的存取，這正是6502 CPU的專長，所以速度自然可以加快許多。

下列的程式可以幫我們將所有HIRES的值算出，並且存到＄5100至＄527F的區間，由於這些HIRES位址我們將會經常使用，因此程式0－2可以存到磁碟片或磁帶中保存。程式0－2執行時間約要等上10秒鐘，可見位址的計算確實費時。

*參照附錄A

```
]LIST

0  REM  *********************
1  REM  * ROUTINE TO CREATE *
2  REM  * LOOK-UP TABLE     *
3  REM  *********************
100  FOR Y = 0 TO 191
200 Y1 =  INT (Y / 8):YR = Y - Y1
      * 8
210 Y2 =  INT (Y1 / 8):YS = Y1 -
     Y2 * 8
220 YL = 8192 + Y2 * 40 + YS * 12
     8 + YR * 1024
230  POKE 20736 + Y, INT (YL / 25
     6)
240  POKE 20928 + Y,YL -  INT (YL
      / 256) * 256
250  NEXT Y
```

程式0－2計算並存放畫面位址表的程式

```
]CALL-151

*5100.527F

5100- 20 24 28 2C 30 34 38 3C
5108- 20 24 28 2C 30 34 38 3C
5110- 21 25 29 2D 31 35 39 3D
5118- 21 25 29 2D 31 35 39 3D
5120- 22 26 2A 2E 32 36 3A 3E
5128- 22 26 2A 2E 32 36 3A 3E
5130- 23 27 2B 2F 33 37 3B 3F
5138- 23 27 2B 2F 33 37 3B 3F
5140- 20 24 28 2C 30 34 38 3C
5148- 20 24 28 2C 30 34 38 3C
5150- 21 25 29 2D 31 35 39 3D
5158- 21 25 29 2D 31 35 39 3D
5160- 22 26 2A 2E 32 36 3A 3E
5168- 22 26 2A 2E 32 36 3A 3E
5170- 23 27 2B 2F 33 37 3B 3F
5178- 23 27 2B 2F 33 37 3B 3F
5180- 20 24 28 2C 30 34 38 3C
5188- 20 24 28 2C 30 34 38 3C
5190- 21 25 29 2D 31 35 39 3D
5198- 21 25 29 2D 31 35 39 3D
51A0- 22 26 2A 2E 32 36 3A 3E
51A8- 22 26 2A 2E 32 36 3A 3E
51B0- 23 27 2B 2F 33 37 3B 3F
51B8- 23 27 2B 2F 33 37 3B 3F
```

由＄5100至＄51BF存放畫面位址表的高位元組

```
51C0- 00 00 00 00 00 00 00 00
51C8- 80 80 80 80 80 80 80 80
51D0- 00 00 00 00 00 00 00 00
51D8- 80 80 80 80 80 80 80 80
51E0- 00 00 00 00 00 00 00 00
51E8- 80 80 80 80 80 80 80 80
51F0- 00 00 00 00 00 00 00 00
51F8- 80 80 80 80 80 80 80 80
5200- 28 28 28 28 28 28 28 28
5208- A8 A8 A8 A8 A8 A8 A8 A8
5210- 28 28 28 28 28 28 28 28
5218- A8 A8 A8 A8 A8 A8 A8 A8
5220- 28 28 28 28 28 28 28 28
5228- A8 A8 A8 A8 A8 A8 A8 A8
5230- 28 28 28 28 28 28 28 28
5238- A8 A8 A8 A8 A8 A8 A8 A8
5240- 50 50 50 50 50 50 50 50
5248- D0 D0 D0 D0 D0 D0 D0 D0
5250- 50 50 50 50 50 50 50 50
5258- D0 D0 D0 D0 D0 D0 D0 D0
5260- 50 50 50 50 50 50 50 50
5268- D0 D0 D0 D0 D0 D0 D0 D0
5270- 50 50 50 50 50 50 50 50
5278- D0 D0 D0 D0 D0 D0 D0 D0
*
```

由＄51C0 至＄527F 存放畫面位址表的低位元組

上面的兩組表格便是由程式0－2所計算出來的每條水平掃描線的最左端點位址。請注意到這兩組表格是分別存放每一位址的高位元組與低位元組，這種方式可以加快資料存取速度，因爲我們只要直接分別對兩個表格取資料便可以了，而不必經由其它手續，而如果不分成兩組表格的話，將會有下圖所示的問題：

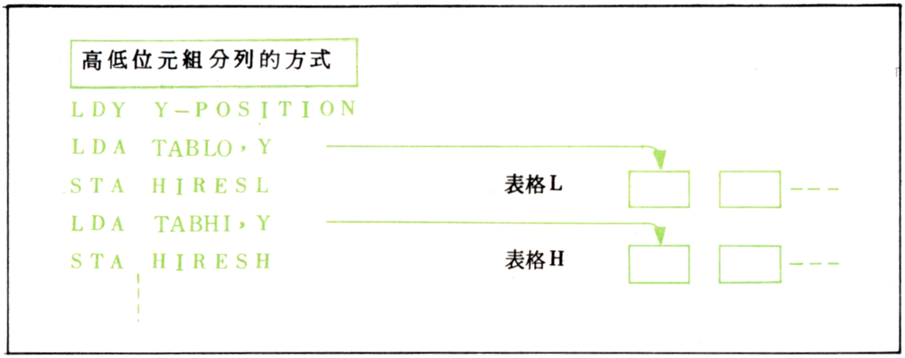


圖 0 － 3 （ a ）

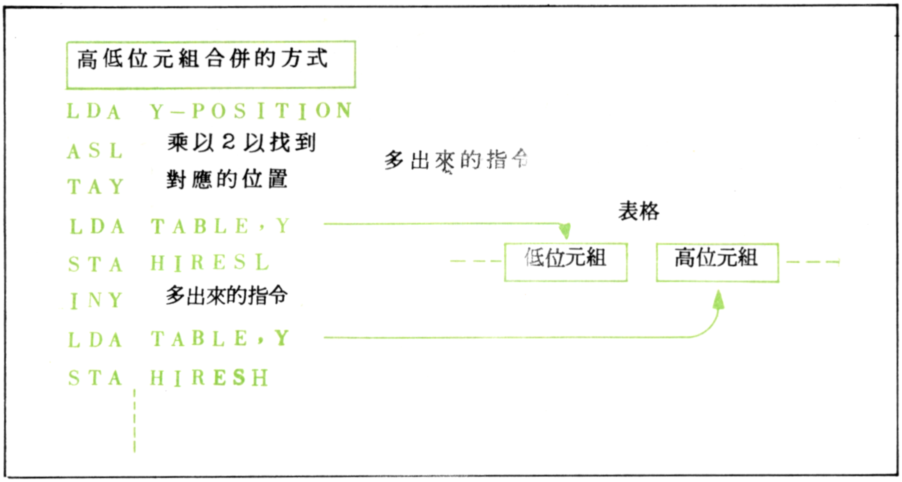


圖 0 － 3 （ b ）

此處，Y － POSITION 是垂直座標值，譬如要取得第三條掃描線的位址，採用高低位元組分列方式，則只要分別到 TABLO+ 3 及 TABHI + 3 的位址去取出第三條掃描線的起始位址便可以了。但是使用高低位元組合併列表的方式，却必須先將座標值存入 A 暫存器（因爲 Y 不能做 ASL ），然後因爲每一起始位址均佔用兩個位元組，所以第三條（從第 0 條算起）掃描線的起始位址是存放在 TABLE ＋ 6 的地方，因此要先 ASL （乘以 2 ）

，再 TAY ，才有辦法在 TABLE + 6 的位置找到第三條掃描線的起始位址的低位元組，最後還要再 INY 才能在 TABLE + 7 的位置取到第 3 掃描線起始位址的高元位組。算算前後的時間，這兩種方式速度差別實在太大了。

0-4 立竿見影的效果

為了要證實我們的理論，請執行下列的程式便可明白。首先請鍵入程式 0 － 3 的機械碼（如果只為了驗證的目的，輸入機械碼最省時間），然後再鍵入 0 － 4 的 BASIC 程式，這個程式與 0 － 1 的程式完全相同，所以我們可以比較 CALL 62454 的 ROM 副程式與 CALL 4096 的自製程式，究竟速度差別有多大！當然，我們是在比較畫面變換時的速度，而不是畫面停留的速度。另外，請不要忘了用程式 0 － 2 來造出高解像畫面的位址表，否則程式是不能動作的。

```
L
;
**********************
*                    *
* SAMPLE PROGRRAM OF *
* LOOK-UP TABLE      *
*                    *
**********************
*
             ORG  $1000
TEMPLO       EPZ  $06
TEMPHI       EPZ  $07
LOOKHI       EQU  $5100
LOOKLO       EQU  $51C0
*
*
```

```
START      STA  $C050
           STA  $C052
           STA  $C057
INIT       LDY  #$0
           LDX  #$0
NEXTLINE   LDA  LOOKLO,Y      ;將（TEMPLO,TEMPHI）
           STA  TEMPLO
           LDA  LOOKHI,Y      ;指向畫面位址表
           STA  TEMPHI
           LDA  #$FF          ;顏色設定
           LDY  #$00
NEXT       STA  (TEMPLO),Y
           INY
           CPY  #$28          ;填完一條水平線？
           BNE  NEXT
           INX
           CPX  #$C0          ;填完整個畫面？
           BEQ  RETURN
           TXA
           TAY
           JMP  NEXTLINE
RETURN     RTS
           END
```

程式 0－3 查表示範副程式

```
*1000.102C

1000-  8D 50 C0 8D 52 C0 8D 57
1008-  C0 A0 00 A2 00 B9 C0 51
1010-  85 06 B9 00 51 85 07 A9
1018-  FF A0 00 91 06 C8 C0 27
1020-  D0 F9 E8 E0 BF F0 05 8A
1028-  A8 4C 0D 10 60
*
```

程式 0－3的機械碼

```
]LIST

0  REM  ********************
1  REM  *** LOOK-UP TABLE ***
2  REM  *** DEMON-PROGRAM ***
3  REM  ********************
10  POKE 4120,122
20  CALL 4096
30  FOR I = 0 TO 500: NEXT I
40  POKE 4120,255
50  CALL 4096
60  FOR J = 0 TO 500: NEXT J
70  GOTO 10
```

程式0－4 快速底色變換程式示範

現在，程式0－4執行起來是否比程式**0－1快**上許多，其效果是否不亞於市售程式的效果？您，已經學到了**一項專業動畫**設計的技巧了。由於 Apple 畫面的結構特殊，以一般方式是無法獲得將畫面由上往下或由下往上做地毯式著色的效果，這也是本程式的特點之一。

比較一下另一種方式

或許您曾經看過比程式0－3更短的程式，譬如程式0－5便是一個例子，但是請明白程式0－3的最大優點是不必經由計算而能夠直接將畫面座標代入運作，程式中29行的CPY #$27（即39）及32行的CPX #$BF（即 191）便是用以限制畫面邊界值的，而程式0－5則是按記憶位址順序來填入圖形值，在應用上說來當然是前者方便多了，因爲後者還需許多手續才可能用來做繪圖工作。至於速度呢？也請比較看看吧！當然程式的戲劇性效果亦是一個重要的考慮因素！

```
********************************
* ANOTHER FILL SCREEN ROUTINE *
********************************
         ORG $1000
HIRESL   EPZ $00
HIRESH   EPZ $01
CLRSCR   LDA #$20          ；從高解像畫面第一點
         STA HIRESL
         TAY               ；開始來着色
         LDA #$00
         STA HIRESH
         TAX
CLR      LDA #$D5          ；偶數位元組着色
         STA (HIRESL),Y
         INY
         LDA #$AA          ；奇數位元組着色
         STA (HIRESL),Y
         INY
         BNE CLR
         INC HIRESH
         DEX               ；畫到底線爲止
         BNE CLR
         RTS
         END
```

程式 0－5 另一種底色變換程式

```
*1000.101B

1000- A9 00 85 00 A8 A9 20 85
1008- 01 AA A9 D5 91 00 C8 A9
1010- AA 91 00 C8 D0 F4 E6 01
1018- CA D0 EF 60
*
```

程式 0－5 的機械碼

```
0  REM *********************
1  REM * BASIC DEMO PRO 0-5 *
2  REM *********************
10  HGR
15  REM ** FILL UP THIS COLOR **

20  POKE 4112,85: POKE 4107,42
25  CALL 4096
30  FOR I = 0 TO 100: NEXT I
35  REM ** FILL UP THIS COLOR **
40  POKE 4112,170: POKE 4107,213
45  CALL 4096
50  FOR J = 0 TO 100: NEXT J
60  GOTO 20
```

程式0－6 另一種底色變換(程式0－5)示範

0-5 所以，怎樣來設計動畫程式呢？

看似奇妙無比的動畫遊戲，其實結構上可說是相當單純，大體上就像圖0－4的流程一般，每個人只要依這個方向一步步的來做，事實上是沒有任何困難的，耐心是一個因素，雖然也有專門輔助動畫設計的程式出現，無論如何，您必須要花下精神心力才會有所結果，當然，這也令我們體會到以兩分鐘拷貝一個程式以外的意義了。

所以，希望您耐心的閱讀下去，我們盡量以有趣的程式及新鮮的題材來保持研習的興趣，但是，此外，自己動手去嘗試設計才是最佳的學習方式，我們盡量提供您各種意見做為參考，而怎樣來設計動畫程式呢？唯有您才能告訴您自己！

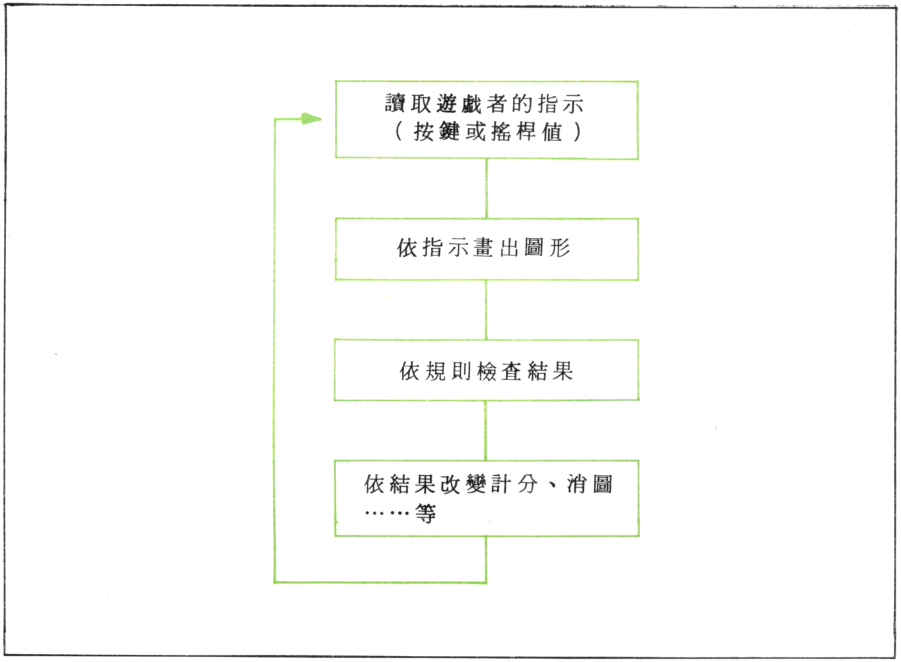


圖 0－4 程式是一個大迴圈

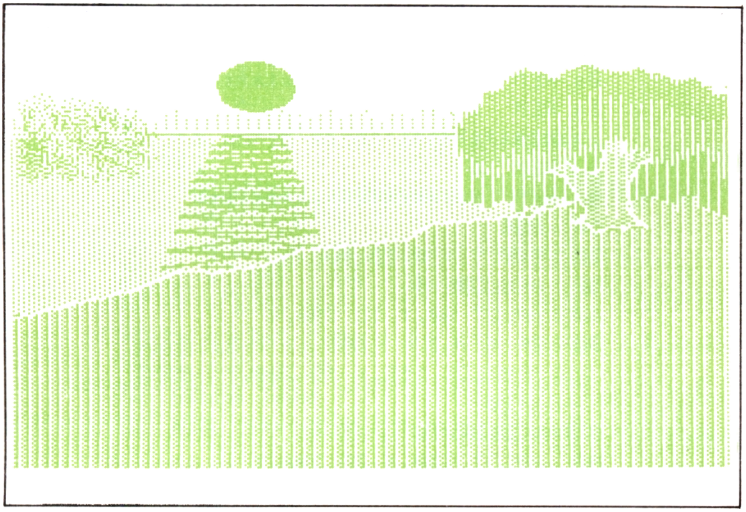
本章提要：

Apple 到底有多少種顏色？許多市售程式的畫面是怎樣造出來的？Apple 手册之外還有什麼呢？

在本章當中，我們要介紹：

1. 無限的 Apple 彩色功能
2. 自動塗色的工具程式
3. 卡通背景的專業技法
4. 各種背景的移動、畫面上捲，繞捲的技巧
5. 以及………？

1 彩色繽紛的蘋果



1-0 APPLE II 只有6種顏色？ 。彩色的徹底研究

我們曾經在一些動畫程式的畫面上，看到“ 好像 ”粉紅色、淺綠色、深綠色、米黃色………等等，原本不屬於 Apple 6 種顏色之列的圖像。這種效果是如何達成的呢？Apple 的手冊不是說：高解像度畫面共有黑、白、橙、藍、黑 1 、白 1 、紫、綠，實質上只有 6 種不同的顏色嗎？那

麼爲什麼會有粉紅、米黃………？

請不要忘記！彩色電視機不是只有紅、綠、藍 3 種顏色的訊號嗎？那麼爲什麼在電視畫面上可以看到黃、紫、橙………？所以，採用相同的原理，我們一樣可以令 Apple 造出各種您希望的彩色效果！

請先嘗試下列的程式：

```
]LIST

0  REM  *******************
1  REM  *                 *
2  REM  * MORE COLOR DEMON *
3  REM  *                 *
4  REM  *******************
10  HGR
20  FOR Y = 0 TO 50
30  HCOLOR= 5
40  HPLOT 80,Y TO 130,Y
80  IF Y / 2 =  INT (Y / 2) THEN
     HPLOT 150,Y TO 200,Y: GOTO
     120
85  HCOLOR= 2
90  HPLOT 150,Y TO 200,Y
100  HPLOT 10,Y TO 60,Y
120  NEXT Y
```

程式1－0 造出奇妙的彩色來

您看到漂亮的粉紅色了吧！（如果顯示的顏色不太相同，那是您的Apple與電視機色相不一致，可以調整電視機的 TINT 或 HUE 使出現紫、橙、粉紅三種顏色便可）所以，巧妙的運用這個性質，你便可以造出各色各樣的美麗圖案了。一切並不神秘，這只不過是在國中時候學到的光的三原色原理而已。下列的程式可以令您體會出一些顏色的基本組合。但是美麗的

顏色不止這些，一切均等待您自己來創造！

```
]LIST

0  REM  ************************
1  REM  *                      *
2  REM  * COLOR! COLOR! COLOR! *
3  REM  *        --BY--        *
4  REM  *        S.U.M.        *
5  REM  *     COPY RIGHT       *
6  REM  *     NOV 18/1982      *
7  REM  *                      *
8  REM  ************************
9  REM
10  HGR :POKE -16302,0
11  FOR I = 0 TO 7
20  FOR J = 0 TO 7
30 X1 =  INT (279 / 8) * I
40 X2 = X1 + 30
50 Y1 =  INT (191 / 8) * J
60 Y2 = Y1 + 20
70  GOSUB 1000
75  REM  *** GO PLOT ROUTINE ***

80  NEXT J
90  NEXT I
100  END
1000  FOR Y = Y1 TO Y2
1005  HCOLOR= I
1010  IF (Y / 2) =  INT (Y / 2) THEN
      HCOLOR= J
1020  HPLOT X1,Y TO X2,Y
1030  NEXT Y
1040  RETURN

]
```

程式1－1　8×8＝64種顏色組合

在程式１－１執行的畫面上，我們可以看到64個方塊，其中當Ｉ＝Ｊ時，是同一顏色混合，並沒有新顏色產生，而Ｉ＝５，Ｊ＝２與Ｉ＝２，Ｊ＝５的結果是一樣的粉紅色，因此共有（36－６）／２＋６＝21種顏色。（爲什麼是36而不是64呢？因爲基本顏色當中，０與４爲黑色，而３與７同爲白色，所以只有６×６＝36個組合。）但顏色並不只限於此，您也可以嘗試用間隔兩種顏色的垂直線造出不同的效果，並且如果是垂直、水平方向均間隔以不同的顏色，則更是奇妙無比了，請參考本書封面圖①、圖②。那麼到底怎樣才是Apple 彩色的極限呢？讓我們繼續往下研究吧！

1-1 讓APPLE來替我們塗色

雖然，上一節中我們畫出了許多前所未見的顏色，但是用來畫圖，尤其是像浮雲、流水、茅屋、樹木等不規則的圖形，幾乎是無法以程式１－１這樣簡單的程式來完成。如果有一個程式，每次我們只要用 Apple 的畫圖筆或是用 HPLOT 來畫出圖形的輪廓，然後一呼叫這個程式，它便能夠自動的替我們將各色各樣的顏色塡入圖框內，那應該是很愉快的事情。所以我們便來設計這樣的程式吧！

我們的想法是，要在圖框內塗色，程式必須自動檢查是否超出邊界，只要程式具有這種能力，那麼我們可以隨便指定圖框內的一個座標值，程式便依這點向上、下、左、右塗色，直到察覺圖框邊界爲止。底下的這個程式，便是一個很有用的塗色副程式，它以＄０，＄１的零頁位址來存放圖框內的任意點的X座標值，以＄２存放Y座標值，顏色值是存放在＄３，而＄４如果是存放＄20，則便是在高解像第一頁畫圖，如果是存入＄40，便必須在第二頁區畫圖。由於程式１－２只是一個機械語言副程式，所以必須由一個主程式來呼叫，程式１－３及１－４便是兩個示範的例子，其執行的結果請參照圖１－０及１－１。程式１－２雖然看來很長，其實其機械碼還不到１Ｋ，所以您可以直接鍵入其機械碼較爲迅速方便，機械

碼的除錯也不難，除了直接對照外，可以用 Apple 的 DISASEMBLER 由 5500L 開始來檢查，除了資料常數之位，若發現？？？則其前後（不一定該位元組本身）的位元組大概是鍵入錯誤了。（註：程式 1 － 2 的著色方式可參考 2 － 3 節）

```
***************************
*                         *
*  HI-RES GRAPHICS PAINTER *
*                         *
*           BY            *
*    WAVE FRONT COMPANY   *
*                         *
***************************
*
*
               ORG $5500
XC             EPZ $0          ；X 座標用兩位元組存放
YC             EPZ $2          ；Y 座標
COLINIT        EPZ $3          ；顏色選擇
HPG            EPZ $4          ；頁區選擇
HIRESL         EPZ $26         ；畫面位址表指位器
HIRESH         EPZ $27
*
*
START          LDA COLINIT     ；選顏色
               ASL
               ASL             ；乘 8 指向顏色表中
               ASL             ；對應的顏色進入點
               STA COLCH
               LDA HPG         ；選頁區
               STA HRPAGE
               LDA YC          ；讀 Y 座標
               STA YS
               LDA XC          ；讀兩位元組的 X 座標
               STA XS          ；（0～279）
               LDY XC+1
```

```
              STY  XS+1
              JSR  HORIZ          ; 取位元組及
              LDA  YC
              JSR  VERT           ; 取 Y 位址
              LDY  BYT0
              LDA  (HIRESL),Y     ; 取所指座標點的
              ASL                 ; 底色
              LSR                 ; 消去N
              BIT  BIT0           ; 是否爲 0
              BEQ  SCN0           ; 何種黑色？
              EOR  #$FF           ; 反相
              BIT  BIT0           ; 反相爲 0 ？
              BEQ  SCN1           ; 何種白色？
              RTS                 ; 都不是則不着色
     SCN0     JSR  SETUPB
              LDX  #0             ; PATTAB=00000000
              LDA  (HIRESL),Y
              BPL  GETPAT         ; 黑 1 ？
              LDX  #1             ; PATTAB=10000000
              BPL  GETPAT         ; 黑 2 ？
     SCN1     JSR  SETUPW
              LDX  #2             ; PATTAB=01111111
              LDA  (HIRESL),Y
              BPL  GETPAT         ; 白1
              LDX  #3             ; PATTAB=11111111
     GETPAT   LDA  PATTAB,X       ; 取PATTAB值
              STA  PATTERN
              LDA  YC             ; 取 Y 座標
     LOOP1    JSR  VERT           ; 開始着色
              LDY  BYT0           ; Y ←位元組
              LDA  (HIRESL),Y     ; 取原值
     INS1     EOR  #$FF           ; 反相
              BIT  BIT0           ; 是否黑或白
              BNE  DONE           ; 不是則不着色
              JSR  FILNE          ; 左右着色一線
              DEC  YC             ; 向上着色
              LDA  YC
```

```
             CMP #$FF         ；頂端？
             BNE LOOP1        ；否，繼續
DONE         LDY YS           ；向上畫完，往下畫
             INY
             STY YC
             TYA
*
*NOW, REPEAT THE ABOVE FOR BELOW
*POINT
*
LOOP2        JSR VERT         ；（同Loop1，往下畫）
             LDY BYT0
             LDA (HIRESL),Y
INS2         EOR #$FF
             BIT BIT0
             BNE DONE1
             JSR FILNE
             INC YC
             LDA YC
             CMP #192
             BNE LOOP2
DONE1        RTS              ；結束主程式
********************************
*                              *
* HORIZ:COMPUTE X OFFSET FOR   *
* THE HIGH-RESOLUTION GRAPHICS *
* SCREEN                       *
*                              *
* A-REG: LSB OF THE X-OFFSET   *
* Y-REG: MSB OF THE X-OFFSET   *
* X RANGE: 0-279($117)         *
*                              *
********************************
HORIZ        PHA              ；暫存A＝XS
             AND #$01         ；奇或偶位址？
             STA EVODD        ；設定奇偶旗號
             LDX #$00
```

```
            TYA             ; 取大於 255 部份，A←XS+1
            BEQ SK256       ; 若小於 256 則到 SK256
            LDY #$04        ; > 256，則 Y 取 4 × 7 = 28
            LDX # 36        ; X 取 36 × 7 = 252
SK256       STY BIT0        ; 暫存Y，BIT0←0 或 4
            CLC
            PLA             ; 取< 256 的部份
            ADC BIT0        ; 加上抵補位元（ 4 或 0 ）
            STA TEMP        ;
            SEC
            SBC #126        ; 扣 18 個位元組（ 7 × 18 = 126)
            BCC NOADJ       ; 若< 126 則到NOADJ
            STA TEMP
            TXA             ; A←0（若大於 126
            SEC             ; 必小於 256 ）
            ADC #17         ; A←18
            TAX             ; X←18
NOADJ       LDA TEMP
            SEC
SUBLOOP     BEQ OUT
            INX             ; 商（由 0 ，18 ，36算起）
            SBC #$07        ; MOD  8
            BCS SUBLOOP     ; 結果> 0 則 Loop
            DEX             ; 由 0 算起，所以X = X − 1
            ADC #$07        ; A = 0～7（餘數）
OUT         STX BYT0        ; 對應的位元組
            TAX
            LDA TABLE,X     ; 取（ 0～7 號）位元罩蓋表
            STA BIT0        ; 最右邊的點
            EOR #$FF        ; 反相
            STA BIT1
            LDA TAB2,X      ; 取對應該點的位元
            STA BITL
            RTS
*
* FILNE-FILL LINE           （著色線）
*
FILNE       STY SBY         ; 暫存水平位元組（ 0～39)
            LDA #$7F
```

```
                STA SMASK       ;SMASK←$ 7F
                LDA BITL        ;取罩蓋
                STA MASK
                LDA (HIRESL),Y
                JSR INS4        ;向左著色
                LDY SBY         ;取回水平位元組
                LDA (HIRESL),Y
INS3            ORA BIT0
                STA (HIRESL),Y
                LDA #$7F
                STA SMASK
                LDA BITL        ;設定顏色罩蓋
                STA MASK
                LSR MASK
                JSR RGT         ;向右著色
                RTS
*
* SETUP
*
SETUPB          LDA NOPINS      ;取'Nop'指令
                STA INS1        ;存入INS1
                STA INS1+1      ;……
                STA INS2        ;……
                STA INS2+1      ;……
                LDA ANDINS      ;取'AND'指令
                STA INS3
                LDA ANDINS+1    ;取BIT1的低位址
                STA INS3+1
                LDA ANDINS+2    ;取BIT1的高位址
                STA INS3+2
                LDA EORINS      ;取'EOR'
                STA INS4
                STA INS5
                LDA EORINS+1    ;取$FF
                STA INS4+1
                STA INS5+1
                RTS
*
* SETUPW        (在白色底色上著色的設定工作)
```

```
*
SETUPW   LDA EORINS        ;取'EOR'指令
         STA INS1
         STA INS2
         LDA EORINS+1      ;取$FF
         STA INS1+1
         STA INS2+1
         LDA ORAINS        ;取'ORA'指令
         STA INS3
         LDA ORAINS+1      ;取BIT0低位元組
         STA INS3+1
         LDA ORAINS+2      ;取BIT0高位元組
         STA INS3+2
         LDA NOPINS        ;取'Nop'指令
         STA INS4
         STA INS5
         STA INS4+1
         STA INS5+1
         RTS
*
NOPINS   NOP
EORINS   EOR #$FF
ANDINS   AND BIT1
ORAINS   ORA BIT0
*
STCOLOR  LDA (HIRESL),Y    ;取對應的資料
         AND SMASK         ;罩蓋掉塗色部份
         STA TEMP2
         LDA SMASK         ;
         CMP #$7F          ;全位元組都不塗色
         BEQ STCOLOUT
         EOR #$FF          ;反相就變成要塗
         STA TEMP3         ;色部份爲1
         LDA YC
         AND #1
         ASL
         ASL
```

```
                STA  TEMP4
                TYA
                AND  #3
                CLC
                ADC  TEMP4
                STA  TEMP4
                LDA  COLCH
                CLC
                ADC  TEMP4
                TAX
                LDA  COLORTAB,X    ; 取顏色值
                STA  COLOR
                LDA  TEMP3         ; 取要塗色位元
                AND  COLOR         ; 着色
                ORA  TEMP2         ; 不着色部份補背景
                STA  (HIRESL),Y    ; 塡顏色
      STCOLOUT  RTS
      *
      * LET-LEFT FILL ROUTINE
      *
      LFT       LDA  (HIRESL),Y    ; 取對應值
                CMP  PATTERN       ; 是否全位元組改變？
                BEQ  CHNGALL       ; 是，則至CHNGALL
      INS4      EOR  #$FF          ; 反相
      LPLFT     BIT  MASK          ; 與MASK比較
                BEQ  DONLFT        ; 相等則至DONLFT
                TAX
                LDA  SMASK         ;
                EOR  MASK          ; 相同則0
                STA  SMASK
                TXA
                LSR  MASK          ; 下一位元
                BNE  LPLFT         ; 尙有其他位元？
                NOP
                BEQ  LFTBK         ; 沒有，至LFTBK
      CHNGALL   LDA  #$00
                STA  SMASK         ; SMASK ← 0
```

（STA TEMP4 至 LDA COLORTAB,X 一段：）依Y座標次序取彩色值，以構成顏色組合效果（顏色值事先由COLOR TAB設好）

*有關STCOLOR可參考2－3節

```
LFTBK    JSR STCOLOR          ;塗顏色
         LDA LBITL            ;由旁邊開始
         STA MASK             ;着色
         LDA #$7F
         STA SMASK            ;SMASK 還原爲$7F
         DEY                  ;尙有多少位元組
         BPL LFT
         RTS
DONLFT   JSR STCOLOR
         RTS
*
*RGT-RIGHT FILL ROUTINE       (與LFT 相似)
*
RGT      LDY SBY
RGTSTRT  LDA (HIRESL),Y
         CMP PATTERN
         BEQ CHGALL2
INS5     EOR #$FF
LPRGT    BIT MASK
         BEQ DONRGT
         TAX
         LDA SMASK
         EOR MASK
         STA SMASK
         TXA
         ASL MASK
         BPL LPRGT
         BMI RGTBK
CHGALL2  LDA #$00
         STA SMASK
RGTBK    JSR STCOLOR
         LDA LBITR
         STA MASK
         LDA #$7F
         STA SMASK
         INY
         CPY #40              ;超過右邊界
```

```
               BNE  RGTSTRT
               RTS
DONRGT         JSR  STCOLOR
               RTS
****************************
*                          *
* HI-RSE GRAPHICS          *
*                          *
* 192 VERTICAL ADDRESS     *
* ($00-$BF)=<abcdefgh>     *
*                          *
*        <abcdefgh>        *
* MAPPED INTO THE          *
* FOLLOWING WORD:          *
*                          *
*<001fghcd> & <eabab000>*
*             OR           *
*<010fghcd> & <eabab000>*
*                          *
*ab value:00,01 & 10       *
*a=1 & b=1 is illegal !    *
*                          *
****************************
* NOW DEFINE HI-RES BASE ADDRESS
*      ( A=b7b6b5b4b3b2b1b0 )
VERT           LDX  #$0        ; 假設 b7 b6 = 0 0
               STA  HIRESL     ; A bit A 等於
               BIT  HIRESL     ; 將 b7 b6 存入 N 及 V 旗號
               BPL  SKX1       ; 若 b7 = 0 則至 SKXI
               LDX  #$A0       ; b7=1, 所以可能 b7b6 =10
SKX1           BVC  SKX2       ; 若 b6=1 則至 SKX2
               LDX  #$50       ; 所以 b7 b6 = 1 0
SKX2           STX  HIRESL     ; HIRESL ← b7b6b7b6 0000
               TAX
               AND  #$7        ; 取座標值的 b0 b1 b2
               ASL
               ASL             ; 乘 4
```

```
                ADC HRPAGE   ; 加上負數
                STA HIRESH
                TXA          ; 取回座標值
                ROR
                ROR
                ROR
                ROR          ; b8 → C
                ROR HIRESL   ; 得到 b8b7b6b7b6 0 0 0
                AND #$3      ; 取 b5b4
                ADC HIRESH   ; 加上 HIRESH
                STA HIRESH   ; 得到 001b2b1b0b5b4
                RTS          ; 或 010b2b1b0b5b4
    EVODD       BYT 0
    BYT0        BYT 0        ; 水平位元組
    BIT0        BYT 0        ; 最右位元的端點處理
    BIT1        BYT 0        ; BIT0 的反相值
    XS          ADR 0
    YS          BYT 0
    SBY         BYT 0
    BITL        BYT 0
    MASK        BYT 0
    SMASK       BYT 0
    HRPAGE      BYT $40
    TEMP        BYT 0
    PATTERN     BYT $FF
    NEWCOLOR    BYT $00
    TAB2        BYT 2        ; 0000010
                BYT 2        ; 0000010
                BYT 4        ; 0000100
                BYT 8        ; 0001000
                BYT 16       ; 0010000
                BYT 32       ; 0100000
    LBITL       BYT 64       ; 1000000
    LBITR       BYT 1        ; 0000001
    TABLE       BYT 3        ; 0000011
```

*有關 VERT 的計算可參考第二冊

```
              BYT 3                  ; 0000011
              BYT 6                  ; 0000110
              BYT 12                 ; 0001100
              BYT 24                 ; 0011000
              BYT 48                 ; 0110000
              BYT 96                 ; 1100000
     PATTAB   BYT 0                  ; 0000000
              BYT 128                ; 1000000
              BYT 127                ; 0111111
              BYT 255                ; 1111111
     TEMP2    BYT 0                  (圖點是倒過來排的)
     TEMP3    BYT 0
     TEMP4    BYT 0
     COLOR    BYT 0
     COLCH    BYT 0
     CPLUS1   BYT 22
     COLORTAB HEX 0000000000000000   ; 黑
              HEX 8080808080808080   ; 黑
              HEX FFFFFFFFFFFFFFFF   ; 白
              HEX 7F7F7F7F7F7F7F7F   ; 白
              HEX 2A552A552A552A55   ; 綠
              HEX AAD5AAD5AAD5AAD5   ; 紅
              HEX 552A552A552A552A   ; 紫
              HEX D5AAD5AAD5AAD5AA   ; 藍
              HEX 33664C194C193366   ; 其他各種
              HEX B3E6CC99CC99B3E6   ; 顏色……
              HEX 2244081108112244   ; …………
              HEX A2C4889188891A2C4
              HEX 1122440844081122
              HEX 91A2C488C48891A2
              HEX 6E5D3B773B776E5D
              HEX EEDDBBF7BBF7EEDD
              HEX 5D3B776E776E5D3B
              HEX DDBBF7EEF7EEDDBB
              HEX 664C19331933664C
              HEX E6CC99B399B3E6CC
```

```
                HEX 33664C194C193366
                HEX B3E6CC99CC99B3E6
    SPCL        HEX FF00FF0000FF00FF   ; 特殊顏色
                END                    ; 由使用者設
                                       ; 定
```

程式1－2 自動着色副程式（請鍵入機械碼並存錄至磁片或磁帶）

```
*5500.5843

5500- A5 03 0A 0A 0A 8D 8A 57
5508- A5 04 8D 6F 57 A5 02 8D
5510- 6A 57 A5 00 8D 68 57 A4
5518- 01 8C 69 57 20 94 55 A5
5520- 02 20 3C 57 AC 65 57 B1
5528- 26 0A 4A 2C 66 57 F0 08
5530- 49 FF 2C 66 57 F0 0E 60
5538- 20 0E 56 A2 00 B1 26 10
5540- 0F A2 01 10 0B 20 42 56
5548- A2 02 B1 26 10 02 A2 03
5550- BD 82 57 8D 71 57 A5 02
5558- 20 3C 57 AC 65 57 B1 26
5560- 49 FF 2C 66 57 D0 0B 20
5568- DF 55 C6 02 A5 02 C9 FF
5570- D0 E6 AC 6A 57 C8 84 02
5578- 98 20 3C 57 AC 65 57 B1
5580- 26 49 FF 2C 66 57 D0 0B
5588- 20 DF 55 E6 02 A5 02 C9
5590- C0 D0 E6 60 48 29 01 8D
5598- 64 57 A2 00 98 F0 04 A0
55A0- 04 A2 36 8C 66 57 18 68
55A8- 6D 66 57 8D 70 57 38 E9
55B0- 7E 90 08 8D 70 57 8A 38
55B8- 69 11 AA AD 70 57 38 F0
55C0- 08 E8 E9 07 B0 F9 CA 69
55C8- 07 8E 65 57 AA BD 7B 57
55D0- 8D 66 57 49 FF 8D 67 57
55D8- BD 73 57 8D 6C 57 60 8C
55E0- 6B 57 A9 7F 8D 6E 57 AD
55E8- 6C 57 8D 6D 57 B1 26 20
55F0- C7 56 AC 6B 57 B1 26 0D
55F8- 66 57 91 26 A9 7F 8D 6E
5600- 57 AD 6C 57 8D 6D 57 4E
5608- 6D 57 20 FC 56 60 AD 76
5610- 56 8D 60 55 8D 61 55 8D
5618- 81 55 8D 82 55 AD 79 56
5620- 8D F7 55 AD 7A 56 8D F8
5628- 55 AD 7B 56 8D F9 55 AD
5630- 77 56 8D C7 56 8D 06 57
5638- AD 78 56 8D C8 56 8D 07
5640- 57 60 AD 77 56 8D 60 55
5648- 8D 81 55 AD 78 56 8D 61
5650- 55 8D 82 55 AD 7C 56 8D
5658- F7 55 AD 7D 56 8D F8 55
5660- AD 7E 56 8D F9 55 AD 76
5668- 56 8D C7 56 8D 06 57 8D
5670- C8 56 8D 07 57 60 EA 49
5678- FF 2D 67 57 0D 66 57 B1
5680- 26 2D 6E 57 8D 86 57 AD
5688- 6E 57 C9 7F F0 31 49 FF
5690- 8D 87 57 A5 02 29 01 0A
5698- 0A 8D 88 57 98 29 03 18
56A0- 6D 88 57 8D 88 57 AD 8A
56A8- 57 18 6D 88 57 AA BD 8C
56B0- 57 8D 89 57 AD 87 57 2D
56B8- 89 57 0D 86 57 91 26 60
56C0- B1 26 CD 71 57 F0 1A 49
56C8- FF 2C 6D 57 F0 2A AA AD
56D0- 6E 57 4D 6D 57 8D 6E 57
56D8- 8A 4E 6D 57 D0 EB EA F0
```

```
56E0- 05 A9 00 8D 6E 57 20 7F
56E8- 56 AD 79 57 8D 6D 57 A9
56F0- 7F 8D 6E 57 88 10 C9 60
56F8- 20 7F 56 60 AC 6B 57 B1
5700- 26 CD 71 57 F0 19 49 FF
5708- 2C 6D 57 F0 2B AA AD 6E
5710- 57 4D 6D 57 8D 6E 57 8A
5718- 0E 6D 57 10 EB 30 05 A9
5720- 00 8D 6E 57 20 7F 56 AD
5728- 7A 57 8D 6D 57 A9 7F 8D
5730- 6E 57 C8 C0 28 D0 C8 60
5738- 20 7F 56 60 A2 00 85 26
5740- 24 26 10 02 A2 A0 50 02
5748- A2 50 86 26 AA 29 07 0A
5750- 0A 6D 6F 57 85 27 8A 6A
5758- 6A 6A 6A 66 26 29 03 65
5760- 27 85 27 60 01 10 0C F3
5768- 73 00 23 10 08 10 73 20
5770- 73 00 00 02 02 04 08 10
5778- 20 40 01 03 03 06 0C 18
5780- 30 60 00 80 7F FF 73 8C
5788- 04 77 80 16 00 00 00 00
5790- 00 00 00 00 80 80 80 80
5798- 80 80 80 80 FF FF FF FF
57A0- FF FF FF FF 7F 7F 7F 7F
57A8- 7F 7F 7F 7F 2A 55 2A 55
57B0- 2A 55 2A 55 AA D5 AA D5
57B8- AA D5 AA D5 55 2A 55 2A
57C0- 55 2A 55 2A D5 AA D5 AA
57C8- D5 AA D5 AA 33 66 4C 19
57D0- 4C 19 33 66 B3 E6 CC 99
57D8- CC 99 B3 E6 22 44 08 11
57E0- 08 11 22 44 A2 C2 88 91
57E8- 88 91 A2 C4 11 22 44 08
57F0- 44 08 11 22 91 A2 C4 88
57F8- C4 88 91 A2 6E 5D 3B 77
5800- 3B 77 6E 5D EE DD BB F7
5808- BB F7 EE DD 5D 3B 77 6E
5810- 77 6E 5D 3B DD BB F7 EE
5818- F7 EE DD BB 66 4C 19 33
5820- 19 33 66 4C E6 CC 99 B3
5828- 99 B3 E6 CC 33 66 4C 19
5830- 4C 19 33 66 B3 E6 CC 99
5838- CC 99 B3 E6 FF 00 FF 00
5840- 00 FF 00 FF
```

程式1-2的機械碼表（請存錄至磁片或卡帶）

```
LIST

0  REM ************************
1  REM *                      *
2  REM *  DEMON PROGRAM FOR   *
3  REM *  MUTICOLOR FILLER    *
4  REM *                      *
5  REM ************************
20 FILIN = 21760: REM  *** $5500
    ***
30 INC = 0.1:PI = 3.14159
40  HGR : HCOLOR= 3
```

```
50 R = 60:CX = 80:CY = 100: GOSUB
     120
60  REM  *** TO DRAW CIRCLE ***
70 CV = 12: GOSUB 220: REM   ***
     FILIN ***
80 R = 60:CX = 150:CY = 100: GOSUB
     120
90 CV = 18: GOSUB 220
95 R = 40:CX = 115:CY = 65: GOSUB
     120
100 CY = 35:CV = 16: GOSUB 220
110  END
120  REM  *** DRAW A CIRCLE ***
130 TH = 0.0
140 X = CX + R *  COS (TH)
150 Y = CY + R *  SIN (TH)
160  HPLOT X,Y
170 TH = TH + INC
180 X = CX + R *  COS (TH):Y = CY
      + R *  SIN (TH)
190  HPLOT  TO X,Y
200  IF TH <  = 2 * PI THEN 170
210  RETURN
220  REM  *** FILL IN COLOR ***
230  POKE 0,CX: POKE 1,0: POKE 2,
     CY: POKE 3,CV: POKE 4,32: CALL
     FILIN: RETURN
```

程式1－3 塗色副程式的使用法示範

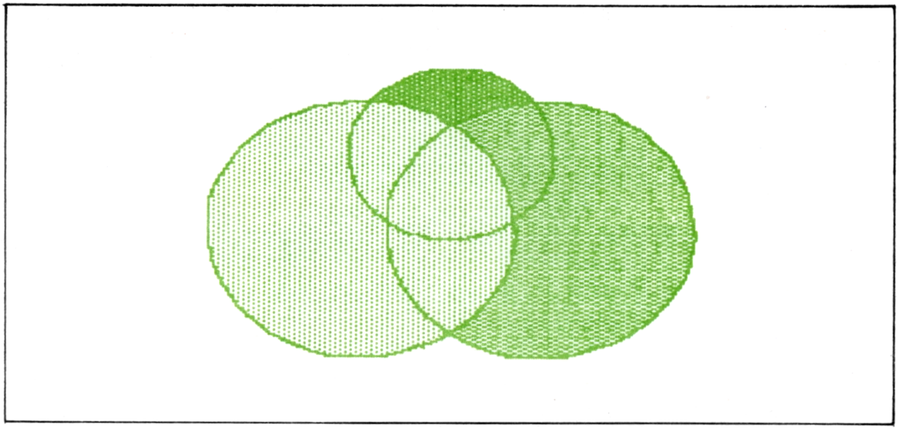


圖 1 — 0

這個程式執行的結果是三個交叉的圓圈，其內部分別塗上了不同的顏色。爲了要更熟習使用 1 — 2 的副程式，我們再來看看底下的這個程式：

```
]LIST

0  REM  ****************
1  REM  *              *
2  REM  * ANOTHER DEMON *
3  REM  *     PROGRAM   *
4  REM  *              *
5  REM  ****************
20  HGR : HCOLOR= 3
30  READ X,Y
40  HPLOT X,Y
50  READ X,Y
60  IF X =  - 1 THEN 180
70  HPLOT  TO X,Y
80  GOTO 50
90  DATA  0,95,63,95,63,80,77,80,
     77,95
```

```
100  DATA  77,80,87,70,90,60,87,5
     0,70,40
110  DATA  53,50,50,60,53,70,63,8
     0
120  DATA  63,115,77,115,77,95,19
     0,95
130  DATA  190,80,197,60,247,60,2
     40,80
140  DATA  190,80,190,105,253,105
     ,253,80
150  DATA  247,60,240,80,253,80,2
     40,80
160  DATA 240,105,253,105,253,95,
     279,95
170  DATA -1,-1
180  READ C,X,Y
190  IF C =  - 1 THEN 280
200  POKE 0,X: POKE 1,0: REM  X-P
     OSITION
202  POKE 2,Y: REM  Y-POSITION
204  POKE 3,C: REM  COLOR-VALUE
206  POKE 4,32: REM  PAGE-1
210  CALL 21760: REM  CALL DRAW-S
     UB
220  GOTO 180
230  DATA  17,135,20,17,30,80,17,
     255,80
240  DATA  10,70,60,4,135,135,4,3
     0,105,4,255,100
250  DATA  18,70,95,13,215,70,7,2
     47,76
260  DATA  8,215,95,8,250,95
270  DATA -1,-1,-1
280  END
```

程式 1－4 卡通的佈景（請存錄至卡帶或磁片）

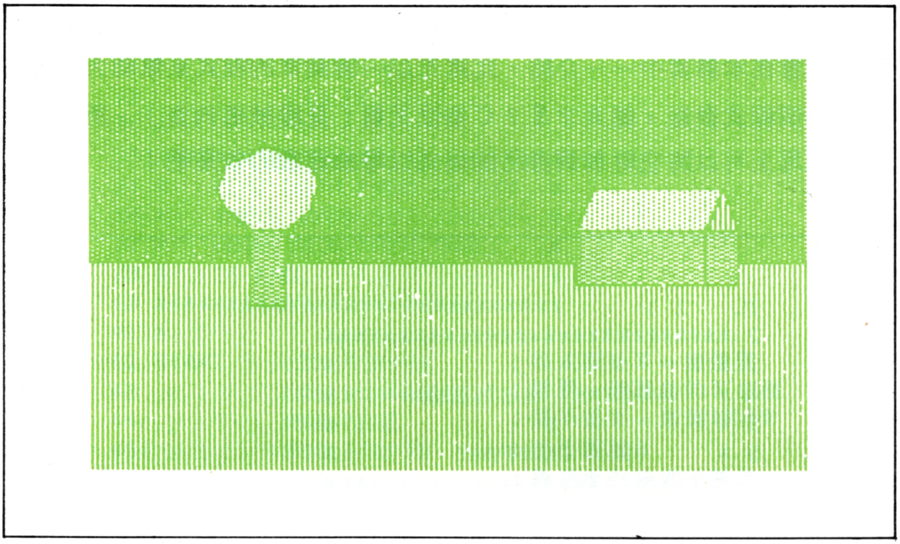


圖1－1

這個程式畫了一顆樹及一間小屋，你可以發現圖中有深綠色的樹葉，綠色的草地，還有淺綠色的牆壁，有深藍色及藍色的屋頂，也有淺藍色的天空，以及米橙色的樹幹。所以您不再說 Apple 高解像畫面的顏色是紫、綠、橙、藍、黑、白………。

而且，您當然會寫一個程式，以按鍵、搖桿、或畫圖版來畫出圖形輪廓，然後移動游標（？）到某一區域，按下" 1 "此區域便塗了紅色，游標移到另外區域，按下" 9 "便塗了水青色，………。

1-2 將背景移動

一個動畫程式的優劣，背景設計往往是一個重要的環節，除了我們在前一節所提到的著色技巧之外，此處我們介紹一個在遊戲程式中常見的背景處理方式，那就是動態的背景。動態背景常見的有賽車道路景觀的移動

、動畫中的小橋流水、星際大戰的銀河星群等等，關於這些背景的設計方法我們會在本書當中一一介紹，此處先以一個簡單的程式來幫助大家瞭解。

這個程式很短，總共才＄7C個位元組（見機械碼列表），所以，說不定您馬上可以將它附加到自己的程式以增加遊戲的效果吧！

```
!L
;
*****************************
*                           *
*    ===START DUST.SUB===   *
*            BY             *
*       W.F CCOMPANY        *
*****************************
*
             ORG $5D00
             OBJ $800
BLACK        EPZ $00
DELTA.Y      EPZ $08        ; 速度存放位址
HCOLOR       EPZ $E4
HPLOT        EQU $F457
LINES        EPZ $A0        ; 垂直範圍最大為160($A0)
KEYBOARD     EQU $C000
KEYREST      EQU $C010
STRPTR       EQU $06        ; DUSTABLE 的指位器
DUSTABLE     EQU $5F80      ; 星群形狀表格起始
WHITE        EQU $FF        ; 位址
YROLL        EQU $07        ; 垂直位置
*
*****************************
*     CLEAR THE GALAXY      *
*****************************
*
             STA $C050      ; 初值設定
             STA $C052
             STA $C057
             LDA #$00
```

```
                STA  STRPTR
                STA  YROLL
                STA  DELTA.Y
*
***************************
*    READ DATA FROM GALAXY   *
*    TABLE WHEN 1ST COORDI-  *
*    NATE IS 0 THEN TABLE    *
*    IS END.                 *
***************************
*
START           LDX  STRPTR          ；程式開始X＝指位器
                LDA  DUSTABLE,X      ；讀第一個星星的X座標
                BNE  MOVE            ；若不是0 ，則到MOVE
                STA  STRPTR          ；指位器歸0
                LDA  YROLL           ；取目前的垂直位置值
                CLC
                ADC  DELTA.Y         ；加上步進值
                CMP  #LINES          ；超過底線？
                BCC  START1          ；若未超過則不減＃LINES
                SBC  #LINES          ；減＃LINES（回到頂端）
START1          STA  YROLL           ；存步進後的Y值
                LDX  #$00
*
***************************
* CHANGE DELTA.Y THIS CAN    *
* BE CHANGE BY PROGRAM IT-   *
* SELF                       *
***************************
*
READKEY         LDA  KEYBOARD        ；讀取鍵盤值
                AND  #$7F            ；化成正常的值
                CLC                  ；清除進位
                ROR                  ；除以2
                STA  DELTA.Y         ；設定速度
                STA  KEYREST         ；還原鍵盤
                JMP  START           ；回到START
*               （主程式結束）
```

```
***************************
* FIND THE OLD POINT AND    *
* TURN IT OFF - PLOT BLACK *
***************************
*
MOVE       PHA                    ;移動副程式，先保存水平座標
           INX                    ;取Y座標
           LDA DUSTABLE,X
           CLC
           ADC YROLL              ;加上目前垂直位置值
           BCC MOVE1              ;若不超過255則不減#LINES
           SBC #LINES             ;減#LINES(減160)
MOVE1      CMP #LINES             ;與#LINES比較
           BCC MOVE2              ;若不超過底線則不減#LINES
           SBC #LINES             ;再減#LINES
MOVE2      TAY                    ;Y座標暫存於Y
           PLA                    ;取用堆疊內容(即X座標)
           PHA                    ;但不影響其狀態
           TAX                    ;置於X
           TYA                    ;A←垂直位置值
           PHA                    ;堆疊←A
           LDY #BLACK             ;Y←0(黑色)
           STY HCOLOR             ;清除目前的點
           JSR HPLOT
*
***************************
* FIND NEW POINT BY OFFSET *
* AND PLOT IT IN WHITE.     *
***************************
*
           PLA                    ;取回垂直座標
           CLC
           ADC DELTA.Y            ;加上步進值
           CMP #LINES             ;是否超過螢幕底線？
           BCC MOVE3              ;否，到MOVE3
           SBC #LINES             ;是，減160
MOVE3      TAY                    ;Y←A
```

```
PLA
TAX
TYA
LDY #WHITE
STY HCOLOR
LDY #$00
JSR HPLOT
INC STRPTR
INC STRPTR
JSR START
END
```

程式1－5 銀河星群副程式

```
5D00.5D7F

5D00- 8D 50 C0 8D 52 C0 8D 57
5D08- C0 A9 00 8D 06 00 8D 07
5D10- 00 85 08 AE 06 00 BD 80
5D18- 5F D0 22 8D 06 00 AD 07
5D20- 00 18 65 08 C9 A0 90 02
5D28- E9 A0 8D 07 00 A2 00 AD
5D30- 00 C0 29 7F 18 6A 85 08
5D38- 8D 10 C0 90 D6 48 E8 BD
5D40- 80 5F 18 6D 07 00 90 02
5D48- E9 A0 C9 A0 90 02 E9 A0
5D50- A8 68 48 AA 98 48 A0 00
5D58- 84 E4 20 57 F4 68 18 65
5D60- 08 C9 A0 90 02 E9 A0 A8
5D68- 68 AA 98 A0 FF 84 E4 A0
5D70- 00 20 57 F4 EE 06 00 EE
5D78- 06 00 20 13 5D 00 20 00
*
```

程式1－5 的機械碼列表

```
1   REM  **************
2   REM  * GALAXY DUST *
3   REM  *  DEMO-PROG  *
4   REM  **************
5   REM
10  FOR P = 24448 TO 24518 STEP 2

20  POKE P, INT ( RND (1) * 255) +
     1
30  POKE P + 1, INT ( RND (1) * 1
     50) + 1
40  NEXT P
50  POKE 24520,0
60  HGR
70  CALL 23808
```

程式1－6 銀河星群的使用法示範

程式1－6是用來說明1－5副程式的使用法，首先必須將程式1－5（或其機械碼）輸入到$5D00～$5D7C之間，然後回到BASIC來鍵入程式1－6。程式1－6執行的時候，您會看到螢幕上有許多細小斑點快速的移動，這時按下CTRL－A 這些點就固定在螢幕上不動，接著按下CTRL－B，這些點便開始緩慢的移動，從ASCII碼的最小值（CTRL－A）到最大值，分別對應星群由慢（不動）到快的各種速度，您也可以寫一個可以用搖桿控制星群上、下移動的程式吧？

程式1-5說明

爲了避免增加程式的複雜性，我們借用了Apple ROM 的HPLOT 副程式（$F457）。不過爲了增加速度起見，一旦我們建立了自己的HPLOT 副程式之後，便應該將$F457修改過來。

HPLOT 副程式：在高解像畫面畫點
進入 HPLOT 之前：A ← 該點的垂直座標值
Y ← 該點的水平座標高位元組部份
X ← 該點的水平座標低位元組部份
$E4 ← 顏色組

圖 1 - 2　HPLOT 副程式的使用法

程式中，爲了簡化起見，Y－暫存器的值我們將它固定爲 0，而 A 的值限制在 0～159 之間，因此 HPLOT 的範圍是垂直 0～159，水平 0～255，當然程式執行時最右邊螢幕區域也就不會有圖形出現了。至於星群的排列情形是取自 $5F80 ～ $5FC8 的表格，程式 1－6 的第10～40行便是以亂數來造出這些表格（也就是決定在何處畫點）。

程式 1－5 當中，由27行～33行是做初值設定，42行～67行是主程式，其中61～67行是鍵盤讀取部份，其中 ROR 是爲了使 CTRL-A 的 ASCII 值變爲 0，以便令畫面靜止。程式中以 $08 的 DELTA,Y 來存放速度值，也就是每次垂直座標增加的值，這個值便是由鍵盤讀入的 ASCII 值除 2（ROR）的結果，所以按不同的鍵便有不同的速度。至於74行～92行的程式部份是用來消去原來的圖點（LDY #BLACK），而99行至115行則是用以畫出新圖點的副程式（LDY #WHITE）依照 HPLOT 副程式的使用法及程式 1－5 的註解，我們是不難明瞭的。

整個程式說來很簡單，首先由亂數造出星群排列，然後程式便依照這個排列在螢幕上畫出圖點來，接著將這些圖點消去，然後將所有點的垂直座標加上按鍵值，並畫出在螢幕上（所以看起來好像往下漂），如此不斷重覆，凡是其垂直座標的值到達螢幕底線的點，便減去 160（即#LINES）而回到螢幕頂端，再往下移動，所以整個畫面看來便有如由太空船往外看到的星群漂浮的景觀。

這個程式的優點在於它並不是掃描整個螢幕（文字幕捲動的方式便是

做全螢幕的掃描），而只是畫出少數幾個點而已。如果是全畫面掃描，則共要 160 × 255 × 2 = 81600 次的畫點動作（本例的圖面是 160 × 255 而不是 192 × 280），現在我們共有35個點，因此只畫了35× 2 =70次而已，此處乘 2 是因為畫圖與消圖的動作。所以我們可以體會到一個快速有效的程式，必須是單一目的的程式，為了移動35個圖點，我們是不需要移動整個高解像畫面上的所有圖點的（請比較下一節的程式速度）。

1-3 垂直與水平的背景移動

畫面垂直下捲

在前一節所提到的背景移動副程式雖然迅速，却只能移動特殊的幾個點而已，如果是像圖 1 — 1 那種複雜的圖形，又怎麼辦呢？下例便是一個可以將畫面做垂直下移的程式：

```
**********************************
* ANOTHER METHODE FOR SCROLLING *
* USES THE LOOK-UP TABLE !!!     *
**********************************
*
         ORG $A00
*
* ZERO-PAGE ADDRESS
*
HIRESL   EPZ $F9
HIRESH   EPZ $FA
DESTIL   EPZ $FB
DESTIH   EPZ $FC
LINE     EPZ $FD
*
* TABLE AND KEY ADDRESS
*
```

```
TABHI      EQU $5100
TABLO      EQU $51C0
KEY        EQU $C000
RESETKEY   EQU $C010
*
* START
*
           STA $C050
           STA $C052
           STA $C057
SCROLL     LDX #$BF              ; X = 191，最底線
LOOP1      LDA TABLO,X           ; 取位址
           STA DESTIL            ; 同時記入 DESTIL
           LDA TABHI,X
           STA DESTIH
           DEX                   ; 向上一線
           LDA TABLO,X           ; 取位址
           STA HIRESL
           LDA TABHI,X
           STA HIRESH
           LDY #$27              ; Y = $27 = 39
LOOP2      LDA (HIRESL),Y        ; 將上一線的最右位元組
           STA (DESTIL),Y        ; 的圖形存入下一線
           DEY                   ; 向左移位元組
           BPL LOOP2             ; 繼續移完一綫
           CPX #$00
           BNE LOOP1             ; 繼續移完畫面
*                                ; (整個畫面向下移一線)
* SCROLL DONE! NOW WE HAVE TO
* BLANK OUT TOP LINE
*
           LDA #$00
           LDY #$27
LOOP3      STA (HIRESL),Y        ; 最上面一線也要
           DEY                   ; 塗黑
           BPL LOOP3
*
* CHECK FOR KEY PRESS
*
```

```
        LDA KEY          ；若按鍵則結束
        BPL SCROLL       ；否則繼續
        BIT RESETKEY
        RTS
```

程式1－7畫面下移程式

```
*A00.A3E

0A00- 8D 50 C0 8D 52 C0 8D 57
0A08- C0 A2 BF BD C0 51 85 FB
0A10- BD 00 51 85 FC CA BD C0
0A18- 51 85 F9 BD 00 51 85 FA
0A20- A0 27 B1 F9 91 FB 88 10
0A28- F9 E0 00 D0 DE A9 00 A0
0A30- 27 91 F9 88 10 FB AD 00
0A38- C0 10 CE 2C 10 C0 60
*
```

程式1－7的機械碼

這個程式令Apple 高解像畫面往下移動，如果您將程式 1 － 2 及 1 － 4 保存起來的話，現在便可以用來造出圖 1 － 1 的背景，並且執行 1 － 7 的程式（ A00G ），是否如所預期的一般？其實畫面上捲或下捲的動作只是記憶區內容的搬動而已，但是要記得，畫面下捲時要由最底下一線開始移動，也就是說將第 191（ #$BF ）號水平線的圖形資料拿掉，然後將第 190 號水平線的圖形資料填到 191號水平線的位址內，依此往上類推便可以了，而最上面一行若填入 # $00時，則畫面下移後就變成黑色了，如果是將第 191 號線的圖形資料填回到第 0 號（最上面一線）水平線的位址內，則便可造出畫面繞捲的效果了。依同樣的方式您也可以造出上捲的動作來。

畫面水平繞捲

那麼如果是水平方向的移動呢？基本的原則是與程式 1 — 7 相似的，只不過這時是每位元組向右（或左）推移而不是向下，所以程式 1 — 8 將畫面最上一線最右一位元組的值取出，然後暫存起來（程式的第32、33行），以便當線上每一位元均向右移一位元組後，可以將最右邊位元組的資料存入最左邊的位址內，這便造出了水平繞捲的效果。

如果您將圖 1 — 1 的畫面以程式 1 — 8 來做水平繞捲（A00G），結果如何呢？由於 Apple 畫面的特殊彩色結構＊，使得同一顏色的點在奇偶位元組的排列位置差異一個位元，所以像程式 1 — 8 這樣一次移動一個位元組（將奇（偶）數位元組的值移入偶（奇）數的位元組內）的方式，便會造成彩色變化的問題了，我們將在以後各章節中陸續的來處理這個問題。

由於程式 1 — 8 並沒有像程式 1 — 7 使用另外一組零頁位址在存放資料動時的目的位址，所以在程式 1 — 8 的34行至39行的 5 行指令當中共使用了 2 次 DEY及 1 次 INY，如果我們在程式最初如程式 1 — 7 一樣的設了一組（DESTIL，DESTIH）指位器，則這些DEY及INY便可以省下來了，由於 LOOP2 （34行至39行）是最內層的廻圈，對執行速度的影響當然很大了，所以請您仿照程式 1 — 7 的方式以（DESTIL，DESTIH）來做目的位址指位器，並觀察是否有任何的影響，以印證我們在 0 — 2 節的說法！當然您也可以仿照本程式將程式 1 — 7 改成繞捲的方式。將程式加以改良或運用是幫助程式瞭解及掌握的好方法。

＊參照附錄 A

```
****************************
* HORIZONTAL SCROLLING ROUTINE  *
* USES THE LOOK-UP TABLE !!!    *
****************************
*
            ORG $A00
*
* ZERO-PAGE ADDRESS
*
HIRESL      EPZ $F9
HIRESH      EPZ $FA
*
* TABLE AND KEY ADDRESS
*
TABHI       EQU $5100
TABLO       EQU $51C0
KEY         EQU $C000
RESETKEY    EQU $C010
*
* START
*
            STA $C050
            STA $C052
            STA $C057
SCROLL      LDX #$00             ；取位址
LOOP1       LDA TABLO,X
            STA HIRESL
            LDA TABHI,X
            STA HIRESH
            LDY #$27             ；暫存最右邊位元組
            LDA (HIRESL),Y       ；的值，以便繞捲
            PHA
LOOP2       DEY                  ； LOOP2 開始右移畫面
            LDA (HIRESL),Y
            INY
            STA (HIRESL),Y
            DEY
```

```
          BNE  LOOP2          ; 移完一線？
          PLA                 ; 繞捲回左
          STA  (HIRESL),Y
          INX                 ; 下一線
          CPX  #$C0           ; 移完 192 條線？
          BNE  LOOP1
*
* CHECK FOR KEY PRESS
*
          LDA  KEY            ; 若按鍵則停止
          BPL  SCROLL
          BIT  RESETKEY
          RTS
          END
```

程式1－8 畫面的水平繞捲

```
*A00.A33

0A00- 8D 50 C0 8D 52 C0 8D 57
0A08- C0 A2 00 BD C0 51 85 F9
0A10- BD 00 51 85 FA A0 27 B1
0A18- F9 48 88 B1 F9 C8 91 F9
0A20- 88 D0 F7 68 91 F9 E8 E0
0A28- C0 D0 E0 AD 00 C0 10 D9
0A30- 2C 10 C0 60
*
```

程式1－8 的機械碼

您是否可以造出一個兩倍或 3 倍於畫面寬度的圖形，然後令它水平繞捲呢？海底城一類的繞捲動作又是如何做的呢？請繼續閱讀………。

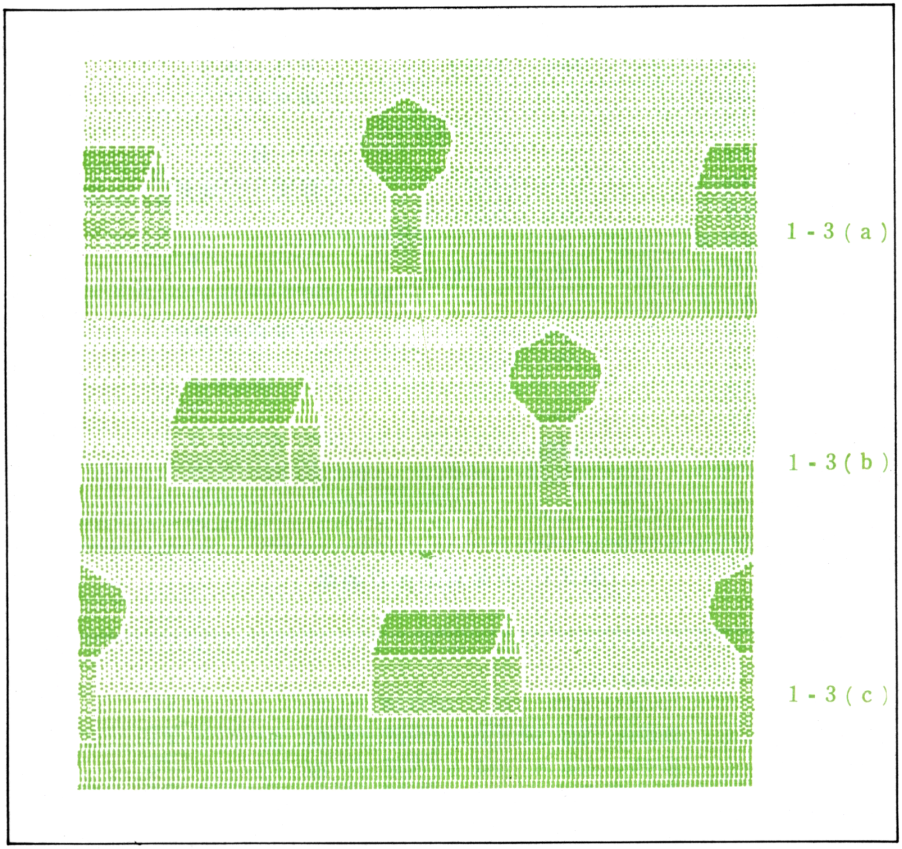


圖 1 - 3　繞捲的 3 幅快照

本章提要

在第一章我們不但學習了彩色背景的繪圖法，並且還知道如何令其移動、繞捲。而畫了故事的背景之後便該是主角登場的時候了，在本章您將學習到：

1. 專業的動畫技巧——預移圖序繪圖法
2. 畫一個小精靈遊戲裡的大嘴巴
3. 彩色圖形移動的問題
4. 繪圖——消圖——背景之間的相互影響與解決方法

2 故事的主角

預移圖序法-----

2-0 畫故事的主角——預移圖序繪圖法

欣賞過很新鮮的彩色技巧之後，接著我們來看看動畫的人物造形又是怎麼畫的呢？你或許會說，我們不是才在1－1節學習到畫圖塗色的方法嗎？動畫人物也是一樣的方法吧！請明白，我們在1－0、1－1節所談到的只是畫面背景（故事的場景）的畫法，這些背景圖案講求的是彩色的

美感，所以我們才花那麼大的工夫專注於彩色效果。可是動畫的主角除了畫出形狀、顏色之外，還要講究其動作的自然與速度，所以這又是另外一段故事了………。

再談速度的問題

一個動畫程式可以在一開始便將場景畫好，所以除非是流動的背景（本書後述），否則速度不是問題。但是所有的角色，却必須在Apple 的畫面上來來往往，速度必然是一個決定性的因素了。

在開始將圖形畫到螢幕之前，讓我們先回顧一下第 0 章所提到的問題：圖形怎樣移動？如果我們畫了一個飛機，而每次以一個機身的距離來移動，也就是說飛機每前進一次，尾巴的位置便移動到原來機頭的前端，如此速度雖然很快，可是對於一個較大的圖形來說，只要移動幾次便已跑完整個螢幕了，這當然不是理想的做法。

那麼，如果移動的距離縮小呢？以最短的距離來說，如果是白色圖形，我們可以沿水平方向一次移動一格，而彩色圖形則要每次兩格來移動，對於這種短距離的移位，當然是不可以用位元組（Byte）來做移動的單位，而是要以每個位元（bit）爲來計算，於是問題便複雜了！想像一個圖形正在向右移，當移到目前所在位元組的第 5 位元時，便要跳過第 6 位元而移到下一位元組的第 0 位元：

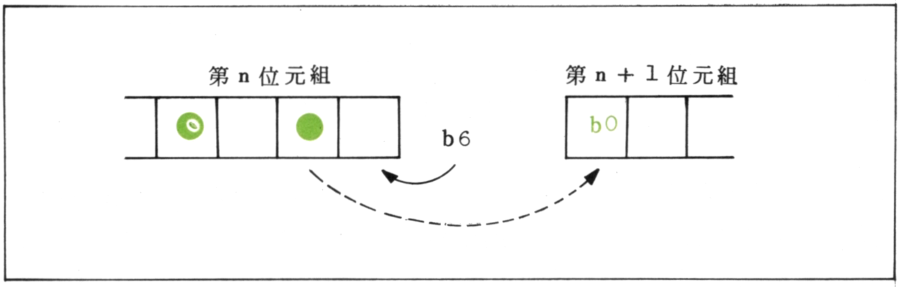


圖 2 － 0

如果是彩色圖形的話，則更要由第 n 位元組的 b 6 跳到第 n ＋ 1 位元組的 b 1 ，或是由第 n 位元組的 b 5 跳到第 n ＋ 1 位元組的 b 0 。對一個稍爲複雜的圖形來說，這是相當煩人也減慢速度的問題。因此，在最新的動畫專業設計當中，多採用一種稱爲預移圖序（ Preshift ）的技巧來解決上述的問題。

像 6502 這種以記憶印射爲導向的CPU，對於記憶單元做存取動作，是其特長之一，速度平均爲 4 ～ 5 個時脈週期。所以，如果螢幕結構是具有週期規則性的話，則一個圖形在螢幕上的運動，只不過是少數幾種形象在週而復始的輪替而已，那麼我們是否可以預先計算好這幾種基本形象的結果（圖形值），然後叫CPU週而復始的來搬動這些圖形到螢幕上的記憶位址內，這就好像我們在一本書的每頁畫上一連串的卡通動作，然後快速翻過所看到的一種視覺效果一般吧！

因此，本節我們就來介紹一下這種對記憶體直接存取的圖形設計法，如果再配合 0 － 3 節的螢幕查表法，則可說是已將 Apple 的畫圖速度發揮到淋漓盡致的境界了。

那麼如何來造出快速的畫圖效果呢？首先，我們必須造出圖形來，圖形的描繪說來是比設計 Applesoft 的圖形表更簡單的事情，我們只要在方格紙上一點一點的將圖形描繪出來便可以了。如果所畫的是黑白圖形，則每一水平位置上的連續兩點是對應到記憶體上的兩個連續位元，如果是彩色圖形，則要依照 Apple 的畫面構造（見附錄A ），間隔一個位元才可填入一點。下圖便是一個簡單的例子，總共只要 4 個位元組 0 E ，1 F ，1 F ，0 E 便將圖形畫出來了。

圖 2 － 1 及 2 － 2 便是兩個圖案的例子，由於 Apple 垂直方向的每點間距比水平方向寬，因此圖案設計時要注意到這個問題而特別將水平方向放大，圖 2 － 1 便是以一個 5 × 4 的點陣來表示一個圓球，而圖 2 － 2 更可以看出設計時水平方向拉大的結果。

高半位元組	低半位元組	低半位元組 2^0	低半位元組 2^1	低半位元組 2^2	低半位元組 2^3	高半位元組 2^0	高半位元組 2^1	高半位元組 2^2	(斜線)	
0	$2^1+2^2+2^3=E$		●	●	●					0 E
$2^0=1$	$2^0+2^1+2^2+2^3=F$	●	●	●	●	●				1 F
$2^0=1$	$2^0+2^1+2^2+2^3=F$	●	●	●	●	●				1 F
0	$2^1+2^2+2^3=E$		●	●	●					0 E

最低位元　　顏色選擇位元

圖 2－1　黑白圖案的設計：一個圓球

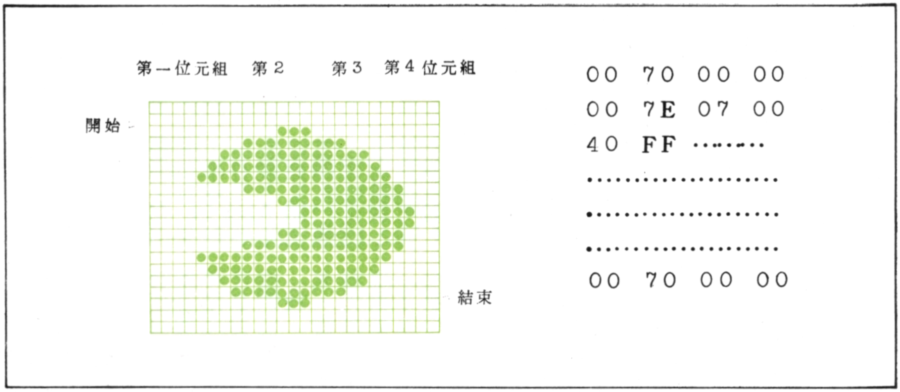


圖 2–2　另一個例子：大嘴巴

2-1 預移圖序法

圖形已經畫了出來，但是怎樣移動呢？以圖 2 － 3 來加以說明吧！如果將小球向右移動一格，則小球的每一圖點必定是移到原位置右邊一格，這時代表小球的 4 個位元組便不再是（ 0 E ，1 F ，1 F ，0 E ）而是（ 1 C ，3 E ，3 E ，1 C ）了，將小球一格一格的向右移，其所對應的 4 個位元組值也相對應的改變，圖 2 － 3 便可說明這個動作。在圖 2 － 3 當

中，小球共移動了連續的 7 個位置，也分別造出了 7 組圖形位元組。如果您繼續將圖 2 － 3 當中的第⑦號圖形再向右移，結果圖 2 － 3 中的 Byte 2 的值便會與第①號圖形 Byte 1 的值完全相同！因此，我們可以說，在整個螢幕運動當中，小球只不過以①～⑦共 7 個面目（圖形值）在輪替出現而已，如果我們將 7 個圖形的值依序存到螢幕位址（Byte1，Byte2）內，則在螢幕上看到的便是〞小球向右移動 7 格〞這樣的動作，這也便是卡通電影的效果了。

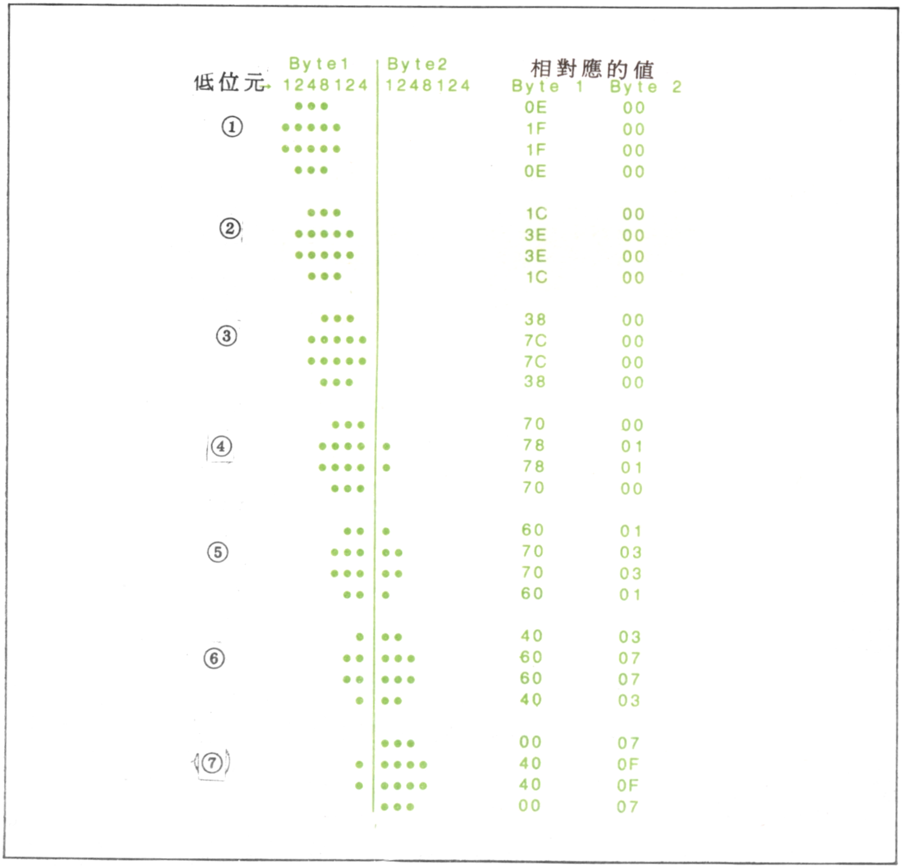


圖 2 － 3　預移圖序表

```
]LIST

1   REM  *********************
2   REM  * BASIC DEMO PROGRAM *
3   REM  * FOR PRE-SHIFT      *
4   REM  * TECHNIQUE          *
5   REM  *********************
10   HGR
20   FOR I = 0 TO 39
40   FOR K = 0 TO 7
50   READ L0,L1,L2,L3,L4,L5,L6,L7
140   POKE 8192 + I,L0: POKE 8193 +
     I,L1
150   POKE 9216 + I,L2: POKE 9217 +
     I,L3
160   POKE 10240 + I,L4: POKE 1024
     1 + I,L5
170   POKE 11264 + I,L6: POKE 1126
     5 + I,L7
200   NEXT K
210   RESTORE
230   NEXT I
300   DATA  14,0,31,0,31,0,14,0,29
     ,0,62,0,62,0,29,0,56,0,124,0
     ,124,0,56,0,112,0,120,1,120,
     1,112,0
310   DATA   96,1,112,3,112,3,96,1
     ,64,3,96,7,96,7,64,3,0,7,64,
     15,64,15,0,7,0,14,0,31,0,31,
     0,14
```

程式 2 - 0　預移圖序的用法

程式 2 — 0 便是預移圖序的簡單說明，我們將圖 2 — 3 的圖序表依次填入到高解像畫面的兩個位址內，由 140行至170 行共把小球的 4 條水平線填完，而40行的 FOR K = 0 TO 7 則把一整套的圖序填完，請注意這時圖序共有 8 個（ 0 ～ 7 ），這是因爲圖 2 — 3 的第⑦圖仍在左位元組留

下兩個點，因此必須再增加再向右移一格的第⑧圖序，才可以將走過的位元組內容完全清除。無論如何，至此我們已證明了預移圖序表的可用性，可是速度是否如預期般的快速呢？在第 4 章我們便會看到以機械語言配合螢幕位址查表法、預移圖預法的實例說明了。

2-2 彩色圖形的水平移動　彩色的預移圖序

由附錄A 我們已經知道了 Apple 畫面的彩色規則，要畫出一個彩色的點，勢必其隔鄰的位置不可以有其它的點，所以一個彩色的圖形畫出來就像圖 2 — 4 一樣。

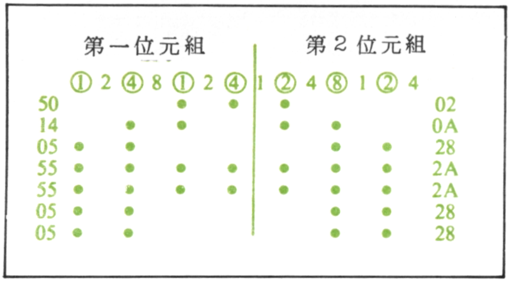


圖 2 - 4　一個 2 位元組寬 × 7 線的外星人

請仔細觀察圖 2 — 4 的點與位元之間的關係，這個圖形的寬度共佔用了兩個位元組，每個位元組的最高位元 b 7 並不直接參與畫圖，因此水平方向共有14點，而間隔畫點的結果便只有我們在圖中所看到的垂直 7 行了。而這 7 行所佔用的位元次序又很特別，在第一位元組當中，它是使用了 b 0、b 2、b 4、b 6 而第二位元組當中則用了 b 1、b 3、b 5，這便是我們在 1 — 3 節中曾提及的彩色繪圖問題。注意到這個特性之後，我們便可以開始來畫彩色圖形的預移圖序表了。

由於 Apple 畫面是相間隔一點才是相同顏色，如果要維持圖形顏色

不變，則一次要水平移動兩點才行。至於每個圖形有幾個圖序呢？每個圖序要佔用幾個位元組呢？如果將 2 — 4 的圖形向右移動，共要移動 7 次後才又回復到原來的位元排列次序（也就是得到相同的圖形資料值），這種情形與 2 — 0 節黑白圖形的圖序值不大相同，前者雖然也是有 7 個圖序，但是只向右移了一個位元組，而現在我們如果只移了一個位元組，則只移了 4 次並且是在圖 2 — 4 第二位元組的第②位元而不是第①位元，因此並沒有恢復到原來的位元次序，必須是如圖 2 — 5 的方式，才是一個完整的圖序週期。

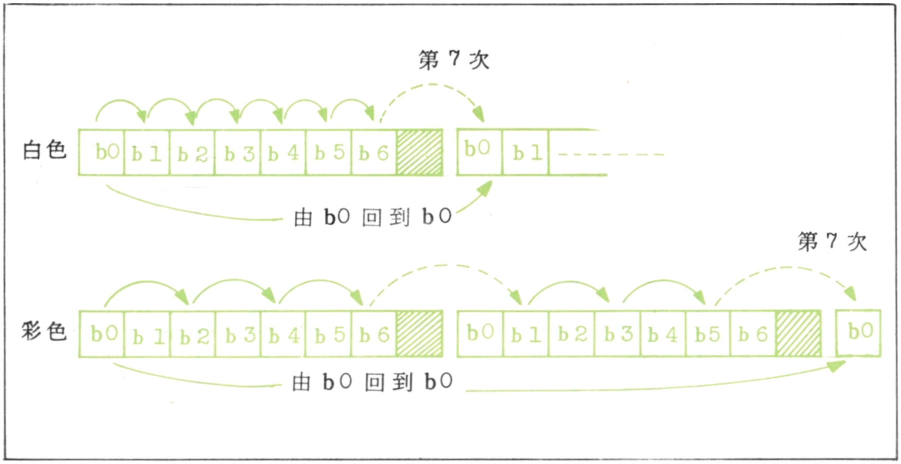


圖 2—5 彩色與黑白圖序的移位

圖 2 — 6 便是彩色的預移圖序，每個圖序的寬度是 4 個位元組而不是像原圖 2 — 4 只有兩個位元組而已。由於彩色圖形每次移兩點，7 個圖序共移了 2 × 7 ＝14 點，所以便要多用兩個位元組了（高位元 b 7 不畫點，所以 2 × 7 ＝14位元）。如果彩色圖形佔用 n 個位元組寬度，則每個圖序寬度共要用 n ＋ 2 個位元組，圖形高度若為 m 條水平線的話，則整個圖序表共要 m ×（ n ＋ 2 ）× 7 個位元組，如果再加上圖序表索引位元組及長寬位元組的話，則還要再加上28個位元組（詳見第 4 章）。

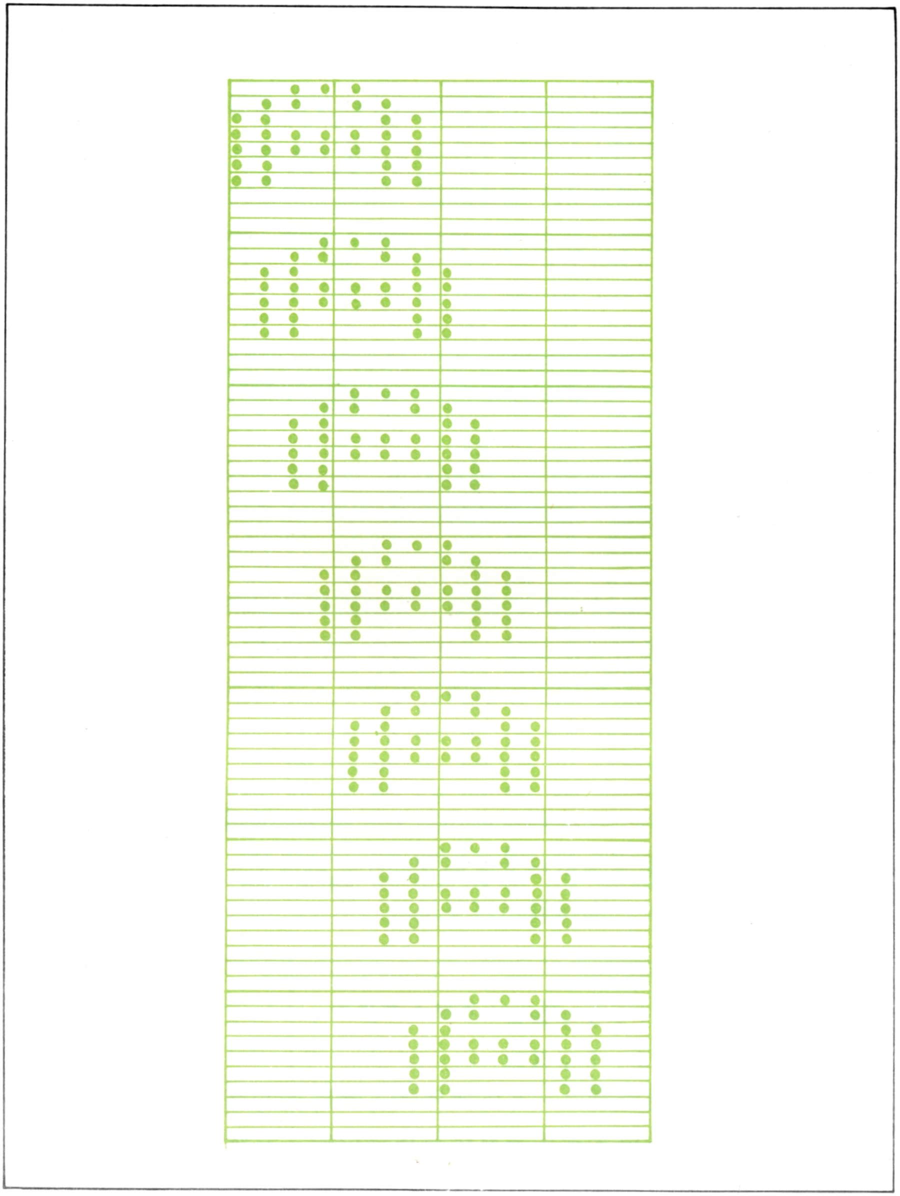


圖 2-6 外星人的彩色預移圖序

```
5300-  00  0E  00  2C  00  4A  00  68
5308-  00  86  00  A4  00  C2  04  07
5310-  50  02  00  00  14  2A  00  00
5318-  05  28  00  00  55  2A  00  00
5320-  55  2A  00  00  05  28  00  00
5328-  05  28  00  00  04  07  40  0A
5330-  00  00  50  28  00  00  14  20
5338-  01  00  54  2A  01  00  54  2A
5340-  01  00  14  20  01  00  14  20
5348-  01  00  04  07  00  2A  00  00
5350-  40  22  01  00  50  00  05  00
5358-  50  2A  05  00  50  2A  05  00
5360-  50  00  05  00  50  00  05  00
5368-  04  07  00  28  01  00  00  0A
5370-  05  00  40  02  24  00  40  2A
5378-  25  00  40  2A  25  00  40  02
5380-  24  00  40  02  24  00  04  07
5388-  00  20  05  00  00  28  14  00
5390-  00  0A  50  00  00  2A  55  00
5398-  00  2A  55  00  00  0A  50  00
53A0-  00  0A  50  00  04  07  00  00
53A8-  15  00  00  20  51  00  00  28
53B0-  40  02  00  28  55  02  00  28
53B8-  55  02  00  28  40  02  00  28
53C0-  40  02  04  07  00  00  54  00
53C8-  00  00  45  02  00  20  01  0A
53D0-  00  20  55  0A  00  20  55  0A
53D8-  00  20  01  0A  00  20  01  0A
*
```

圖 2–7 外星人圖序表

圖 2 － 7 的表格是圖 2 － 6 的圖序表，圖序表的寫法是由圖形最上面一線開始，將圖點依位元次序換算成爲16進位值，當然，我們知道低位元是對應到左邊的點，然後往右排過來的，所以最上面一線的 4 個位元組（圖寬 4 位元組）的值爲50，02，00，00，依此類推便可以將圖序表列出來

了。爲了以後程式的使用，我們在每個圖序的開頭加入圖形的長寬值，以目前的例子寬爲 4 位元組，長爲 7 位元組，所以每個圖序前頭加入04　07，而在總圖序表的最前頭又加上每個圖序相對於前頭的位址距離，即 0 E，2 C，4 A，68，86，A 4，C 2 共 7 個值，由於考慮到較大圖形的情況，距離可能超過255 個位元組，因此共用了兩個位元組來存此資料，這便是圖 2 － 7 的由來，在第 4 章我們將看到更進一步改良的方法。

2 3　畫了圖形之後——各種繪圖法的檢討

動畫與一般畫圖不同之處，在於必須不斷的在新位置畫出新的圖形，而且，請別忘記！也要塗去舊位置上的圖形才行！而且要走得不留一點痕跡（當然迷魂車除外）。

可是，我們使用的是機械語言，並沒有 Applesoft 的 HCOLOR＝0 或 XDRAW 等指令，所以對於這個問題，就必須好好的考慮一下了（尤其是多重彩色背景的時候！）。現在將幾種常用的技巧介紹如下：

(一) EOR　畫圖法

EOR是一個有趣的邏輯運算，它的運算原則是：任何兩個相同的運算元EOR的結果爲 0 ，任何兩個不同的運算元EOR的結果爲 1 。但是邏輯運算元其實只有 0 跟 1 而已，所以其結果可以歸納成：

運算元 A － 1 0	1	0
運算元 B － 1 0	0	1
結果：　　0 0	1	1

圖 2－8　(a)　EOR 的眞值表

運算元 A － 1 0	1	0
運算元 B － 1 1	0	0
0 1 （結果 A 反相）	1	0 （結果 A 不變）

圖 2 － 8 (b)　另一種 EOR 的看法

在圖 2 — 8 的兩個結果當中，(a)就是依照上述基本原則推導得到的，而圖(b)則是計算上較方便的一種看法，這種方法是：任何元素與 1 運算，結果反相，任何元素與 0 運算，結果不變。

所以如果畫面背景是黑色的話，一切就很好辦，只要每次畫圖時，與所欲畫圖位址的內含值做EOR，便可達到塗去舊圖形的目的了。因為要畫新圖形時，畫面底色是黑色（ 0000000 ），任何數與 0 EOR結果不變，所以就畫了一個新圖形。當要消去舊圖形時，只要拿原圖形來做EOR運算，其結果自然是（ 0000000 ）了，這是因為圖形值相同，而依照EOR的原理，相同的值EOR結果為 0 。

```
EOR  B      A←A⊻B（EOR的結果存於A）

          0011010 ——— 新圖形   }
     EOR：0000000 ——— 黑色底色 }   (1)
     ------------------------
          0011010 ——— 結果不變 }
     EOR：0011010 ——— 原圖形   }   (2)
     ------------------------
          0000000 ——— 圖形消失
```

*此處我們是取圖形當中的一條水平線來做說明

圖 2—9（a）EOR畫圖法

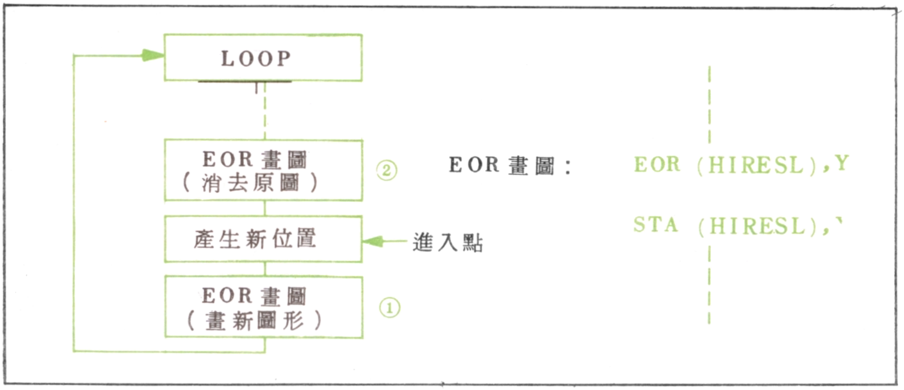


圖 2 — 9（b） EOR畫圖流程

由圖2－9可以看出，由進入點開始，第一次EOR畫圖時，因該位置之底色為黑色，不影響圖形，所以便畫出一個圖形來。而經由LOOP回來之後，是仍舊在該位置做EOR運算畫圖，因此便與先前圖形重疊而消失了。必須等到產生新位置後，才又畫出下一個圖形來。這便是使用EOR來畫圖的基本原理了，在圖2－9中要注意到的是舊圖形消去與新圖形產生的時間間隔極短，而新圖形畫好之後却要經由LOOP及其它過程，才做消去原圖的動作，這是為了使圖形在畫面上存在的時間比消失的時間更長，以避免圖形閃動的情形發生。

對於黑色背景的畫面（諸如小蜜蜂的背景或小精靈（Pacman）的路徑均是），上述的方法就足夠使圖形自然的移動了。可是，如果是彩色的背景呢？請考慮下面的例子：

```
      0111110 —— 新圖形（白色直線）  ┐
EOR : 0001000 —— 背景底色            ┘ (1)
--------------------------------------------
      0110110 —— 結果改變（白線當中產生黑點）┐
EOR : 0111110 —— 原圖形                    ┘ (2)
--------------------------------------------
      0001000 —— 背景還原
```

圖2－10　彩色背景的EOR

結果我們發現，背景不受影響，但是所要畫的圖形却產生了不必要的黑點。使用EOR方式來畫圖，可以保持背景不受破壞，而且因動作簡單所以速度最快，但是當圖形通過彩色背景時，圖形本身多少會產生不必要的斑點缺陷。可是儘管如此，一些Game譬如Bandits或Tsunami均因其快速簡潔而採用此法。採用這種方法的最大一個缺點是，兩個完全相同的物體相遇重疊時，因為EOR的作用，而會完全消失（正常情況是仍會看到一個圖像），必須等到交叉過去後，才又出現圖樣。經由這樣的體認之後，您是否可以僅僅由一個Game的畫面的觀察，就可以不必研究其程式而可

以指出其畫圖的方法呢？

(二) ORA 畫圖法

6502 的或運算（OR）是將A——暫存器的內含值與被運算元做或運算，因此稱為ORA。其運算原則是：除了0 OR 0的結果為0之外，其它運算的結果均為1。

運算元A —	0	1	0	1
運算元B —	0	0	1	1
結果：	0	1	1	1

ORA B　　A←A OR B

圖 2−11(a) 或運算眞值表

現在來觀察將ORA指令用在畫圖的情形：

	圖形（前4位）
暫存器A的值（圖形資料）——	1 1 1 1 0 0 0
背景資料 ——	1 0 1 0 1 0 1
結　果：	1 1 1 1 1 0 1
	圖形（前4位）　背景（共存）（後3位）

圖 2−11 (b) 圖形與背景重疊後的情形

由圖 2 −11我們可以發現到，圖形與背影相重疊的結果，雖然看似已將暫存器A的值改變了，其實這是圖形長度只有4個位元，而其它3個為0的位元是透明的（請注意透明就是黑色），這是ORA運算的一個特色，所以結果是圖形與背景共存，相互的干擾減少了(與EOR不同)，如此我們便可以減低前述EOR時，圖形受彩色背景干擾的缺點了。

可是背景圖案呢？是否可以像EOR一般的經過兩次EOR就可以還原呢？圖 2 −12證明這是不可能的：

	1 1 1 1 0 0 0	—— 圖形資料（存於 A ）
ORA：	1 0 1 0 1 0 1	—— 背景資料
	1 1 1 1 1 0 1	—— 結果共存於畫面
ORA：	1 1 1 1 0 0 0	—— 圖形資料
	1 1 1 1 1 0 1	—— 結果背景無法還原

* 以一條水平線做說明

圖 2 —12　彩色背景的ＯＲ運算

解決這個問題的方法是，當要清除舊圖形時，只要將原來背景圖案的資料（值）存入舊圖形所在位址，便可以將舊圖形蓋過而且恢復背景舊觀了。可是背景資料在哪裡呢？不是已經與圖形做ORA運算而改變無法恢復了嗎？一個方法是每次要畫圖做ORA運算時，便先將背景資料保存起來，當要恢復舊觀時再取回來用。可是有一個更快速而方便的方法，那便是利用Apple的第 2 個畫面（如果您的程式用到兩個頁區交替變化，那麼可以用第 3 、 4 頁區，第 3 、 4 頁區？），每次程式一開始，做背景畫面初值設定時，便同時也在第二頁區畫一個相同的背景圖形，這個背景圖形並不是用來畫圖的，而是當您的第一頁區的背景因ORA而受圖形覆蓋而無法在恢復舊觀時取回原資料 ，我們便可在第 2 頁區相對的位址上（加上＄2000）找到相同的背景資料來塡回到第 1 頁區，一切便很完美了。這種方法，因爲不必先保留背景資料存到其它地方，因此速度上自然加快了（不過仍然比EOR慢）。

一個可以用的ＲＯＭ副程式：＄ＦＣＡ８我們一直強調要使用快速的專用程式，而不要使用ＡＰＰＬＥ　ＲＯＭ的副程式，可是＄ＦＣＡ８不但是ＲＯＭ的副程式，而且是延時副程式，為什麼要使用它呢？請參考圖 2 － 9 的説明，每次畫圖之後，均要

以一個延時迴路來使圖形存在螢幕的時間久一點，所以便要使用延時程式了。既然是延時程式，也就不要求快速的執行時間，因此＄FCA8是一個可以用的ROM副程式！有一個觀念必須清楚，那就是所謂速度，是指畫圖與消圖的動作過程要快速（很多BASIC的程式，畫圖動作看得很清楚，那便是速度的問題），可是畫完圖之後，却不一定急着消去圖形，所以怎樣適當的控制程式的快慢緩急，是要花工夫來體會的。

㈢AND 畫圖法

為什麼要考慮這麼多的畫圖法呢？因為我們用的是機械語言，我們只有LDA、STA、EOR、AND、…………。

EOR與 OR 的方式，雖然快速簡單，可是在彩色背景的狀況下，却不免會因為圖形與背景相互混合，而影響畫面效果。歸根究底的說來，這些問題的來源，皆因為背景彩色的緣故，如果，將要畫圖的部份先清除成黑色，然後再畫圖，一切問題不就解決了嗎？所以只要將所要畫圖的位置填入0………。

問題是，圖形寬度不一定是佔滿整數個位元組，如果圖形寬度是一個半的位元組，而我們將0存入到所要畫圖的位址（兩個位元組），則在圖形的左右必然會產生黑色的邊緣輪廓。交互使用AND、OR 、EOR便可以解決上述的問題而畫出美妙的圖案效果來。

首先我們要將畫圖位置的背景影塗黑，但又不可影響其前後邊緣。在最初設計圖案時，要同時也畫一個相同形狀的白色圖形，用這個白色圖形我們才可以定出詳細的圖形位置：

第一位元組	第二位元組	
1 0 1 1 0 1 1	0 1 0 1 0 0 1	背景
1 1 0 1 0 0 1	0 1 0 0 0 0 0	原圖形
1 1 1 1 1 1 1	1 1 0 0 0 0 0	相同大小的白色圖形

圖形大小（1 1 1 1 1 1 1 1 1）

圖 2 -13 (a)

1 1 1 1 1 1 1 1 1 0 0 0 0 0	白色圖形
1 1 1 1 1 1 1 1 1 1 1 1 1 1	EOR #$FF
0 0 0 0 0 0 0 0 0 1 1 1 1 1	造出黑色區域
1 0 1 1 0 1 1 0 1 0 1 0 0 1	AND 背景
0 0 0 0 0 0 0 0 0 0 1 0 0 1	這時才可畫圖

畫圖位置（0 0 0 0 0 0 0 0 0）　邊緣不受影響（0 1 0 0 1）

圖 2 -13 (b)

0 0 0 0 0 0 0 0 0 0 1 0 0 1	黑色區域＋背景
1 1 0 1 0 0 1 0 1 0 0 0 0 0	ORA 原圖形
1 1 0 1 0 0 1 0 1 0 1 0 0 1	

圖形（1 1 0 1 0 0 1 0 1）　背景（0 1 0 0 1）

圖 2 -13 (c)

由圖 2 —13 (c)我們可以看出，圖形與背景是眞正的各自分開互不混合了。至於要恢復原來背景的景觀，可以用前述的第二頁區方式或背景資料暫存方式便可以了。Beer Run 這個 Game ，便是使用這種方法來造出其特別清晰明朗的畫面。

(四)直接畫圖法

有些場合，我們只需要在螢幕的某一部份移動物體而不是整個畫面，這時我們可以將整個區域（通常爲1/8～1/16畫面）的圖形重新畫過，也就是說直接以新畫面來將舊畫面蓋過。如果是黑色背景，則雖然是在整個畫面移動，這種方法也很好用，因爲它不必做EOR、AND、………等動作，只要用相同大小的黑色方塊將先前的圖形蓋住便可以了，Space Eggs及Gorgon 便是使用這種方法的 Game 。我們將在第 4 章看到它的使用實例，這種方法的最大優點是不會造成畫面閃動的現象，並且由於是直接繪圖，因此速度比EOR的方式更快。不過在設計預移圖序及其圖序表時要注意在圖形上下左右留下一行空白（即黑色）的輪廓，如此才有辦法將先前的圖形完全蓋過（若是每次移動兩格的彩色圖形，則輪廓寬度要佔兩行才行），有關更詳盡的實例及說明，請見 4 — 0 節。

各種繪圖方法的比較

	速度	圖形佔用空間	碰撞檢知	閃爍	注意事項
EOR	中等	高解像第 1 頁區或第 2 頁區（共 8 K位元組）	能檢知與其他圖形的碰撞	有	無特殊限制
AND 並保留底色	中等	高解像第 1 頁區或第 2 頁區＋背景資料緩衝區	〃	有	碰撞後的圖形必須特別處理

直接繪圖保留背景	快速	高解像第 1 及第 2 頁區（共 16K 位元組）	〃	無	圖形必須加上黑色（高位元為 0，即 0 0）的輪廓來遮沒移位間距
直接繪圖	快速	高解像第 1 或第 2 頁區（共（8 K 位元組）	無	無	圖形必須加上底色輪廓來遮沒移位間距

2-4 碰撞的問題

在一些動態的場合裡，常需要做碰撞檢查，基本上，這種碰撞檢知，其實就是對畫面位址內含值的檢查，只要所欲畫圖位址的內含值與背景底色的值不同，則此位址必然是已經有其它的物體存在了，此刻，如果再畫上任何圖形，碰撞便發生了。（令我們同時想到的是，我們是否也可以用同樣的方法來避免碰撞發生呢？）

那麼，如何來檢知碰撞呢？雖然在所欲畫圖的位元組內已經有其它物體（圖形）存在，可是却不一定就發生碰撞，我們可以由下圖來觀察：

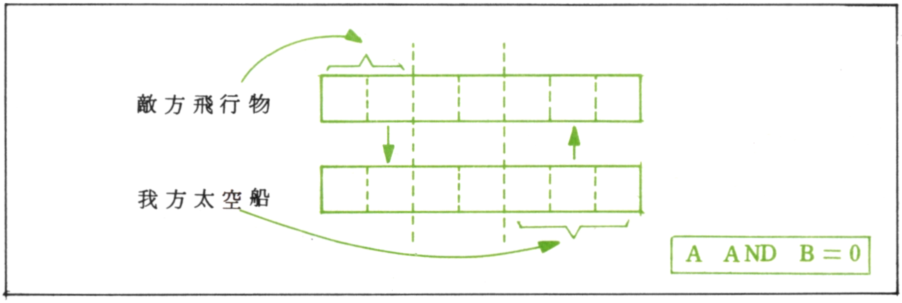


圖 2 - 14 雖然是在同一位元組，但却不一定碰撞：A AND B = 0

所以要檢知碰撞並不是只檢查位元組內容即可，像圖 2 —14 一般，兩個物體事實上並沒有眞正的碰撞。為了確保眞正的碰撞，除了檢知畫面位址的內含值之外，還要拿我們所要畫的圖形與之比較才行，我們的比較法

是：

```
              LDA   SHAPE,X        ;取圖形值
              STA   TEMP           ;暫存原值
              AND   (HIRESL),Y     ;是否碰撞
              BEQ   CONTINUE       ;否，繼續
              STA   COLLIS         ;是，設定碰撞旗號
                      ⋮
CONTINUE      LDA   TEMP
                      ⋮
```

程式 2 - 1 碰撞檢知法

程式2－1當中，SHAPE是由在第4章所將述及的自動位址暫定法所設定的圖形表位址，所以程式首先取出圖形值，暫存原值後，便與畫面位址的內含值做AND運算，這樣，圖2－14所描述的狀況便不會誤當成碰撞了，因爲這時AND的結果是0，所以BEQ到CONTINUE繼續執行程式。如果眞正發生了碰撞，則AND結果不爲0，於是便將這個不爲0的結果存入COLLIS旗號中。爲什麼要製定一個碰撞旗號呢？爲什麼不直接就摧毀被碰撞的物體呢？因爲通常碰撞檢知是在繪圖的同時執行的，也就是包含於繪圖副程式內，但是碰撞之後要如何處置呢？可能碰撞的結果是加10分，也可能加20分或扣5分，甚至是整艘太空船爆炸，而這些動作則應該是由主程式統籌處理才是合理的，所以我們只在碰撞時設定旗號，記錄下碰撞事件，而等到回到主程式才加以處理，另外一個理由是使用EOR繪圖時，必須整個圖形畫完後才可消圖，否則會有殘缺的圖形留在畫面上。

可是上述的方法並不十分完整，首先是最高位元的問題，由於畫面位址的最高位元並不直接參與畫圖，所以雖然（$80）AND（$AA)結果不爲0，但$80代表的是黑色的背景，所以這種情形也不能當成碰撞，解決的辦法是AND（$7F），也就是說令最高位元爲0（即最高位元忽視不計。）

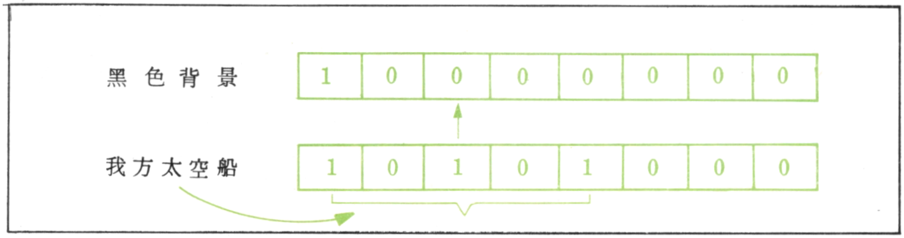


圖2-15 AND結果不爲0，但不碰撞

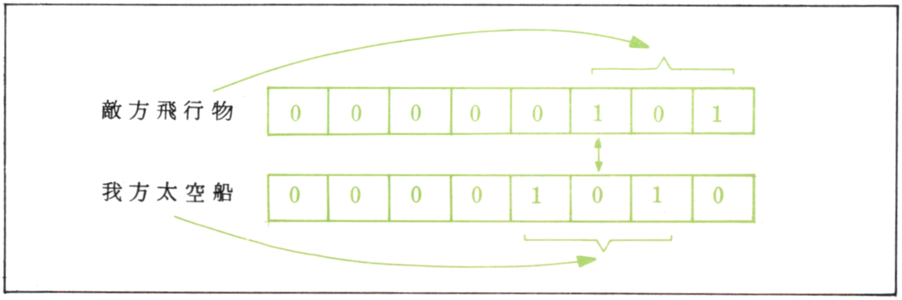


圖2-16 AND結果爲0，但事實上却發生碰撞

對於這種情形，由於彩色圖形是0、1相間，不同顏色的圖形可能恰好0、1相叉，AND運算的結果爲0，但事實上已經是碰撞了。解決這個問題的方法是在做碰撞檢知時，將其中一個彩色圖形代之以白色圖形，如此一來，便不會有0、1相間的情形了。至於眞正畫圖時還是要用彩色圖形才行（白色圖形只做碰撞檢知用）。

```
          LDA   WHITE,X        ;取白色圖形
          AND   #$7F           ;去掉最高位元的影響
          AND   (HIRESL),Y     ;與畫面內含值比較
          BEQ   CONTINUE       ;若是0，則繼續
          STA   COLLIS
                  :
CONTINUE  LDA   SHAPE,X        ;取眞正的彩色圖形
                  :            ;並且開始畫圖
```

程式2-2　完整的碰撞檢知方式

本章提要

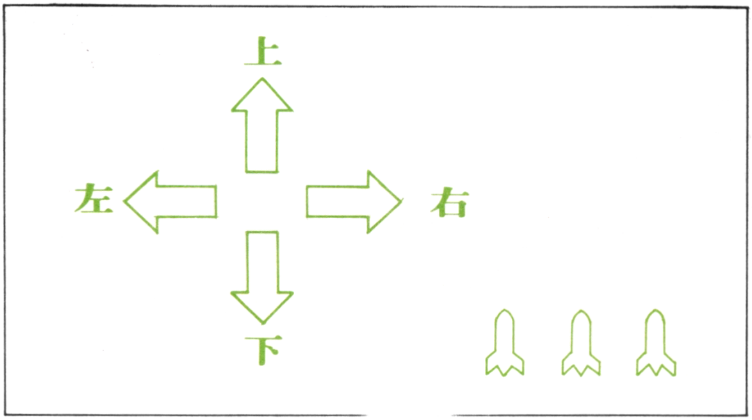
到目前爲止，我們學習了美麗的彩色繪圖，也知道什麼叫預移圖序，並且檢討了如何使用EOR、ORA、AND來畫圖的基本理論。所以，進一步，我們便來看看這些技法如何付諸實現！

在第三章，您將學習到：

1. 將畫面位址查表法付諸實現：GETADR副程式
2. 將EOR繪圖法付諸實現：XDRAW 副程式
3. 令您控制全局遊戲的：READKEY 、PADDLE副程式
4. 徹底瞭解搖桿問題的癥結
5. 彩色背景下的 EOR、ORA、AND繪圖法
6. 彩色物體的水平移動

最後，您將可以自己寫一個純粹由機械語言構成的動畫程式！

3 ARCADE GAME的構造



3-0 使用畫面位址表：GETADR副程式

第一章的彩色畫圖雖然動人又好玩，可是那只不過是背景而已，接下來要做的，便是要將故事的主角搬上螢幕。這是一個較複雜的工作。我們將在本章當中將每一個枝節部門予以說明，然後組合成一個完整的程式。雖然是一些枝節細部，却也是學習動畫設計的基礎訓練，所以希望您耐心

的跟我們一齊研究下去！

首先，我們來學習如何使用畫面位址表，爲了循序漸進起見，讓我們從最簡單的問題開始。假設我們畫了一個圖形，譬如是圖 A-3 或 2 － 2 ，要令這個圖形在畫面上移動，最好的方法是有一個副程式，每次我們將畫面位置輸入，它便能回給我們對應的畫面位址，如此只要接著將圖形內容（一位元組的資料）存入此位址，圖形便在畫面上出現了。當然，這樣子我們最多只不過是畫了水平的 7 個點而已，不過只要寫個迴圈，便不難將整個圖形畫出來，馬上我們便可看到工作是怎麼進行的。下例便是我們所要的一個畫面位址查表的副程式：

```
**************************
*    GETADR ROUTINE !!!   *
* INPUT VRRIABLES :       *
* VERT (IN Y-REGISTER)    *
* & HORIZ                 *
* RETURN:(HIRESL,HIRESH)  *
**************************
*
* GETADR ROUTINE
*
GETADR    LDA TABLO ,Y
          CLC
          ADC HORIZ
          STA HIRESL
          LDA TABHI ,Y
          STA HIRESH
          RTS
**************************
* END OF GETADR ROUTINE   *
**************************
```

程式 3 - 0　畫面位址查表副程式例

這個程式必須配合程式 0 － 2 的造表程式來使用，也就是必須以 0 － 2 的 BASIC 程式造出畫面位址表，其高低位址表的起始位址分別定為 TABHI 及 TABLO ，然後程式 3 － 0 便依我們所輸入的畫面垂直水平座標（VERT，HORIZ）的值來查出 TABHI 及 TABLO 表中所對應的位址，然後存入（HIRESL，HIRESH），這便是GETADR 的工作了，其中畫面垂直座標值VERT 是事先在主程式中已存入 Y －－暫存器內，至於 GETADR 的用途我們將在底下的程式中加以說明：

3-1 繪圖程式：XDRAW副程式

```
*******************************
* XDRAW ROUTINE SAMPLE !!!     *
* INPUT VARIABLES: HORIZ, VERT *
*******************************
*
XDRAW      LDY VERT          ；輸入 VERT 至 Y
           JSR GETADR        ；查出位址
           LDX #$00
           LDY #$00
XDRAW2     LDA (SHPL,X)      ；開始畫一位元組的
           EOR (HIRESL),Y    ；圖形
           STA (HIRESL),Y
           INC SHPL          ；向右移一位元組
           INY
           DEC LNGH          ；還有多少寬度未畫？
           BNE XDRAW2        ；未完？
           INC VERT          ；指向下一線
           DEC DEPTH         ；還有多少高度未畫？
           BNE XDRAW         ；全部畫完？
           RTS
*****************************
* END OF XDRAW ROUTINE !!! *
*****************************
```

程式 3 － 1　DRAW 畫圖－擦圖副程式

我們已知道大部份的動畫繪圖，均以斥或運算（ Exclusive OR ）來處理，斥或運算的特性是經過兩次斥或執行之後，圖形完全消除而底色或背景則毫不影響的恢復原狀。所以 XDRAW 第一次在畫面上畫出圖形之後，若在原位置上再 XDRAW 一次，則圖形便從畫面消失，因此 XDRAW 可以當成畫圖副程式，也可以當成擦圖的副程式。

所以，程式 3 — 1 性質上是與 Applesoft 的XDRAW 指令一樣性質的，但是Applesoft XDRAW 的缺點是除了速度慢之外，也很難細緻的畫出多重彩色的實心圖形。程式 3 — 1 的畫圖方式是將水平方向畫圖的迴圈放到垂直方向迴圈之內，這樣做有一個好處是每次在垂直迴圈取得一次位址之後（叫用 GETADR），整個水平迴圈便可以不必再呼叫GETADR，如果兩個迴圈次序對調的話，則每畫一線便要叫用一次 GETADR（因爲垂直座標已改變），執行時間便徒然浪費許多了。

使用程式 3 — 1時，要先造好畫面位址表及程式 3 — 0 ，並且圖形資料也要事先存入，並且以（ SHPL ， SHPH ）來存放其起始位址。

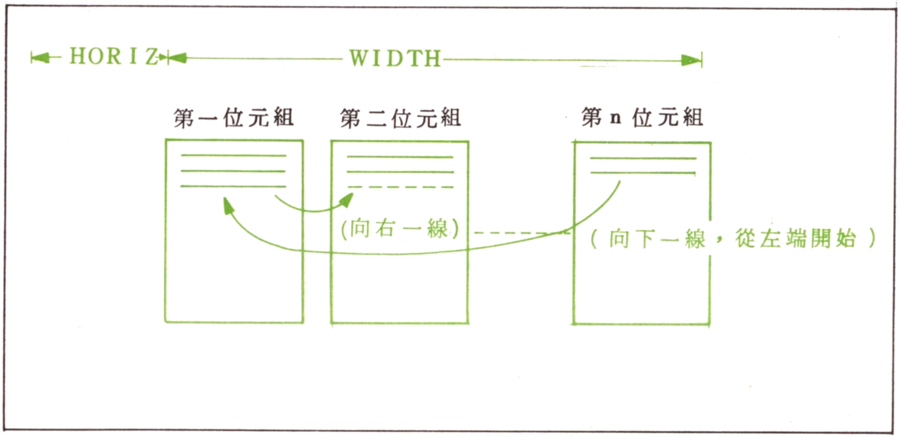


圖 3 - 0 以程式 3 - 1 畫圖的過程

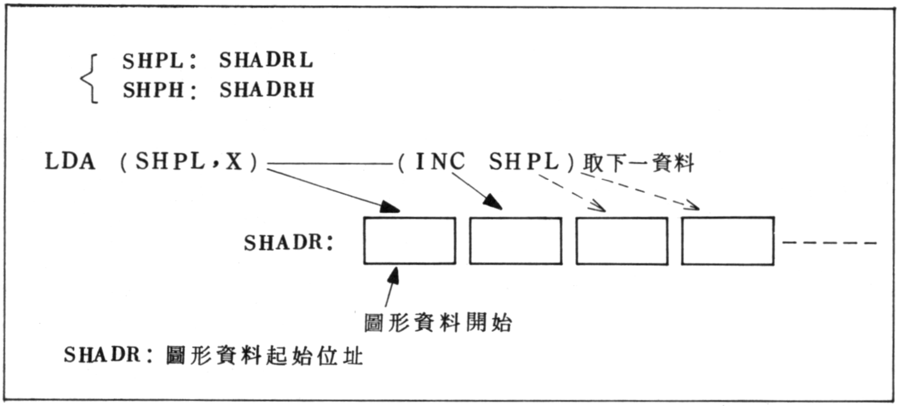


圖3－1 以LDA (SHPL, X) 取圖形資料的方法

如果將圖3－2的火箭圖形資料配合程式3－0，3－1來使用便可以造出很基本的動畫來。

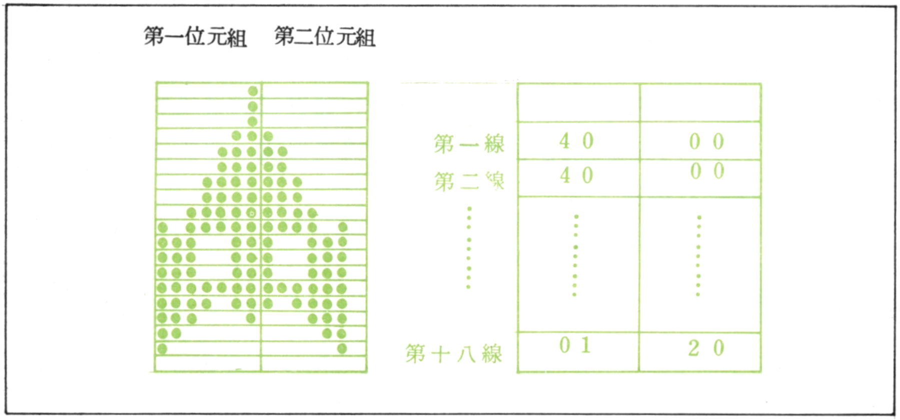


圖3－2 火箭圖形及其圖形值換算法

由於 XDRAW 的畫圖方式是依圖3－0的方式來進行的，因此圖形資料的排列方式是水平兩個位元組排列完畢之後再往下一水平線繼續排列，圖3－2的圖形資料表便是依此方式排列的。

```
*1180.11A3

1180- 40 00 40 00 40 00 60 01
1188- 70 03 70 03 78 07 78 07
1190- 7C 0F 7D 2F 67 39 67 39
1198- 67 39 7F 3F 6F 3D 47 38
11A0- 03 30 01 20
*

SHADR=$1180
```

圖 3－3 圖形表

3-2 讀鍵程式：READKEY

我們已經知道如何使用畫面位址表來畫圖，使用 GETADR及 XDRAW 我們馬上可以造出一個動畫程式來，但是任何動畫程式必須可以與使用者交通才行，所以，現在便來看看如何來設計這些程式，才能使我們畫出的各個角色，操縱於指掌之間。

A.Z.←、→

```
2 HORIZ      DFS  1
3 VERT       DFS  1
4 KEY        EQU  $C000
5 *
6 * MAIN ROUTINE
7 *
```

```
* BEGIN HERE
*.........
*.........
*
READKEY    LDA KEY       ;讀鍵
           CMP #$95      ;'→'?
           BEQ RIGHT
           CMP #$88      ;'←'?
           BEQ LEFT
           CMP #$C1      ;'A'?
           BEQ UP
           CMP #$DA      ;'Z'?
           BEQ DOWN
           JMP BEGIN
*=====================END OF MAIN ***
LEFT       LDA HORIZ
           CMP #$00      ;左端?
           BEQ BEGIN
           DEC HORIZ     ;水平減一(向左)
           JMP BEGIN
RIGHT      LDA HORIZ
           CMP #$27      ;右端?
           BEQ BEGIN
           INC HORIZ     ;水平加一(向右)
           JMP BEGIN
UP         LDA VERT
           CMP #$00      ;頂端?
           BEQ BEGIN
           DEC VERT      ;垂直減一(向上)
           JMP BEGIN
DOWN       LDA VERT
           CMP #$BF      ;底線?
           BEQ BEGIN
           INC VERT      ;垂直加一(向上)
           JMP BEGIN
*------------END OF KEY ROUTINE
```

程式 3-2 讀鍵的程式

程式３－２的結構很簡單，其中要特別注意的是第12行的LDA KEY，KEY的位址是＄C000，如果我們將這個位址的值取入A－－暫存器，便可以讀取按鍵的最新資料，但是如果在下次再讀取按鍵值之前，沒有按下新鍵，則永遠會讀到相同的數值，這是因爲Apple 鍵盤的特性：最後一次按鍵值永遠會保存到下次按鍵之前。利用這種特性，我們可以很方便的來設計程式。

觀察程式３－２可以發現 HORIZ 及 VERT 是我們在程式３－０ GETADR用來查位址及程式３－１ XDRAW 用來畫圖形的兩個變數，它們分別代表所欲畫的圖形最左上端的座標位置（即最左上端位元組的座標位置）。程式３－２的第23行至42行便是改變這兩個變數的值來控制圖形上、下、左、右的移動。而第12行至21行的READKEY 是用來檢知按鍵值。

一個有趣的現象是，只要按下一個鍵，譬如〃→〃，則HORIZ 便加一（即向右移動一個位元組），但是在下次按鍵之前，程式永遠是讀到〃→〃的值，因此按下〃→〃鍵並不是只有令圖形向右移動一個位置，在您下次按鍵之前，圖形會繼續向右移，同樣的，按下〃A〃之後圖形便一直往上移動了。

3-3 搖桿的問題：READPDL

我們接著來考慮搖桿控制的問題，用搖桿來控制物體移動與鍵盤控制有何不同呢？其最大的不同是搖桿是由可變電阻造成的（有一種是由４個開關所組成，其動作與按鍵方式相似），其數值可以連續性的變化，問題便在於我們如何將搖桿的這種變化來與物體位置相連繫起來呢？

首先回到程式３－２來觀察，由23行至42行可以發現到，原來程式是以檢知按鍵結果來將X－Y座標值做對應的加或減一個單位（INC或DEC）。可是搖桿的情況並不相同，是否我們可以直接以搖桿值（PDL函數值由０～255）來設定X－Y座標呢？果眞如此的話，則當我們將搖桿由左

擺到右時，所控制的物體必然立刻要馬上移動到水平座標的最右端了，這樣做有兩個缺點，第一是如果程式的畫圖速度不夠快，則便會產生間斷畫圖的現象，也就是說當程式在某一位置畫完圖之後，搖桿值已做大幅度變化，而使得程式必須馬上到目前搖桿值所指定的位置畫圖，這樣相繼的兩個圖形之間必然會發生不連繫的動作，相信初設計 Game 的人必然有這種經驗。第二是即使程式速度夠快，但由於程式的需要，必需製造某些阻尼效果（譬如小精靈便是），在這些情況下都必須對搖桿的動作原理詳加研究才能造出完美的效果來。

APPLE怎樣讀取PDL？

原廠的Apple 搖桿是由 150K 的可變電阻構成的，當搖桿插入到 Apple 的 Game I／O 插座時，便與主機板內的一個 0.02μf 的電容器並聯成爲 NE558 的RC網路，通常這個RC網路是被＋5 V的電壓充電成爲〃1〃狀態，如果我們將這個RC網路放電爲〃0〃，然後計算其恢復到 1 所要花費的時間，這個時間是與RC值成正比的，C是固定爲0.02 μf ，所以網路恢復時間便與R（也就是搖桿位置）成正比了。

要怎樣才可以令RC網路放電呢？只要觸發＄C070（ PTRIG ）這個位址便可以了，圖 3 － 5 便可說明這個動作。至於眞正的Apple 線路是以一個 NE558 的 4 組振盪器按排成爲單擊電路（ one-shot ）來做爲解碼器到RC網路間的緩衝以獲得更佳的效果。由於NE558 的介入，使得RC網路的電位被反相，因此觸發時，Apple 讀到的是〃1〃而RC恢復時，Apple 讀到的是〃0〃。

程式 3 － 3 便是Apple 的搖桿讀取程式，首先第 2 行（在ROM裡頭爲＄FBIE ）將PDL觸發成 1（RC值爲 0 ），然後設定計數器Y的值爲 0，第 6 行讀取由X所指示的該PDL的值，若其值大於 128（Bit7 ＝1 ）則 INY，若Y不大於 255，則回到第 6 行再讀PDL值，如此直到單擊電路恢復到128以下（Bit7 ＝0 ），這時Y所計數到的值便是PDL值了，

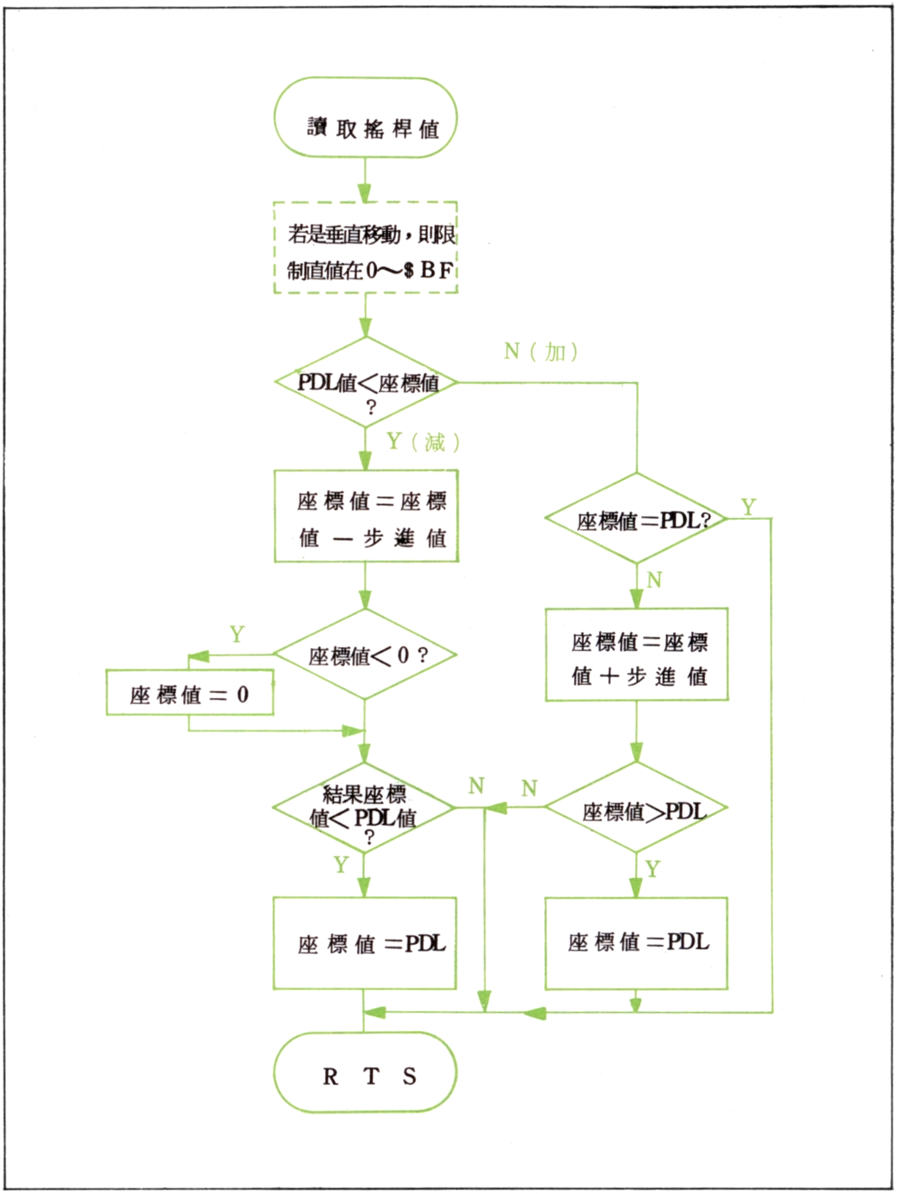

讀取搖桿值
若是垂直移動，則限制值在0～$BF
PDL值<座標值?
N(加)
Y(減)
座標值=座標值－步進值
座標值=PDL?
Y
N
座標值=座標值＋步進值
座標值<0?
Y
座標值=0
結果座標值<PDL值?
N
N
座標值>PDL
Y
Y
座標值=PDL
座標值=PDL
RTS

圖 3-4 搖桿副程式流程

若R值大，則電路恢復慢所以PDL值大，反之則PDL值小，由於第 9 行及第10行的限制，使得讀到的Y 值不大於 255 ，這便是爲什麼PDL的值是在 0 ～ 255 之間而不是畫面寬度 0 ～ 279 的原因了。至於爲什麼以 128 作爲判斷電路狀態的值，是因爲558 單擊電路的輸出是接到H 11 IC的11脚，這是DATA BUS 的D7 也就是Bit7 。

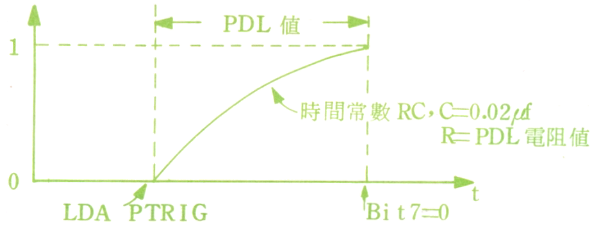


圖 3 - 5　　觸發 PDL

```
 2 PREAD     LDA PTRIG      ; 觸發 NE558
 3           LDY #$00
 4           NOP
 5           NOP
 6 PREAD2    LDA PADDL0,X   ; 第 X 號 PDL 是否< 128 ?
 7           BPL RTS2D
 8           INY            ; 記錄時間＝搖桿值
 9           BNE PREAD2
10           DEY
11 RTS2D     RTS
```

程式 3 - 3　APPLE 監督程式的PREAD

由圖 3 － 4 的流程說明可以看出，原來搖桿值只是做爲物體位置的一個參考比較，而不是直接用來設定位置。每次物體眞正的位置若與搖桿不同（即與操縱者預期位置不同），則便以程式預先設定的一個步進值（即每次

移動的距離）向搖桿值趨近，所以每次執行一遍搖桿程式，物體才靠近一個步進值而已，而這個步進距離當然是在程式中安排好的。因此也就不會有上述間斷畫圖的缺點了（距離移動間隔過大）。。

你可能會發現，這種做法不是很像按鍵程式嗎？只不過前者是以搖桿值來代替A、Z、←、→，而搖桿却又是較爲人所偏愛的一種操縱方式，所以如何將軟體、硬體、機械、人體工學各方面做合理妥善的安排，是未來電腦應用上的一大課題。

搖桿的串音問題

串音（ Crosstalk ）這個字原本指的是立體音響左右聲音相互之間的干擾，在遊戲搖桿設計上，有時候是必須以兩組可變電阻來控制上、下、左、右的移動，這時您會發現事情並不容易………。

由於Apple 是使用NE558IC來做搖桿值的讀取，也就是說，以不同可變電阻的數值來振盪出不同的週期，如果週期與電阻值成正比的話，電阻值便可用來代表PDL的值了。問題是558包含4個獨立的振盪器，但Apple却將4個振盪器的觸發控制連成一點而統一由$C070這個位址來控制，也就是說每次讀取其中一個PDL值，另外其它三組振盪器也一齊振盪，而當我們以快速的機械語言來讀另一個PDL值時，振盪器便來不及歸零而讀到不正確的結果了。所以Apple PDL的串音問題是在於4個振盪器同時被觸發，觸其中一個，便使另外3個PDL值也受影響了。

要解決這樣的問題，我們可以在讀取第一個PDL值之後，經過一段時間等到所有振盪器歸零後再予觸發讀取，這通常必須約200μs的時間，在這個時間內我們可以令程式處理其它的工作。另外一個方法是兩個PDL值同時讀取，也就是不等第1個PDL值讀完再讀第2個，而是同時讀，到此您可能有點迷惑，請參考下圖當可明白：

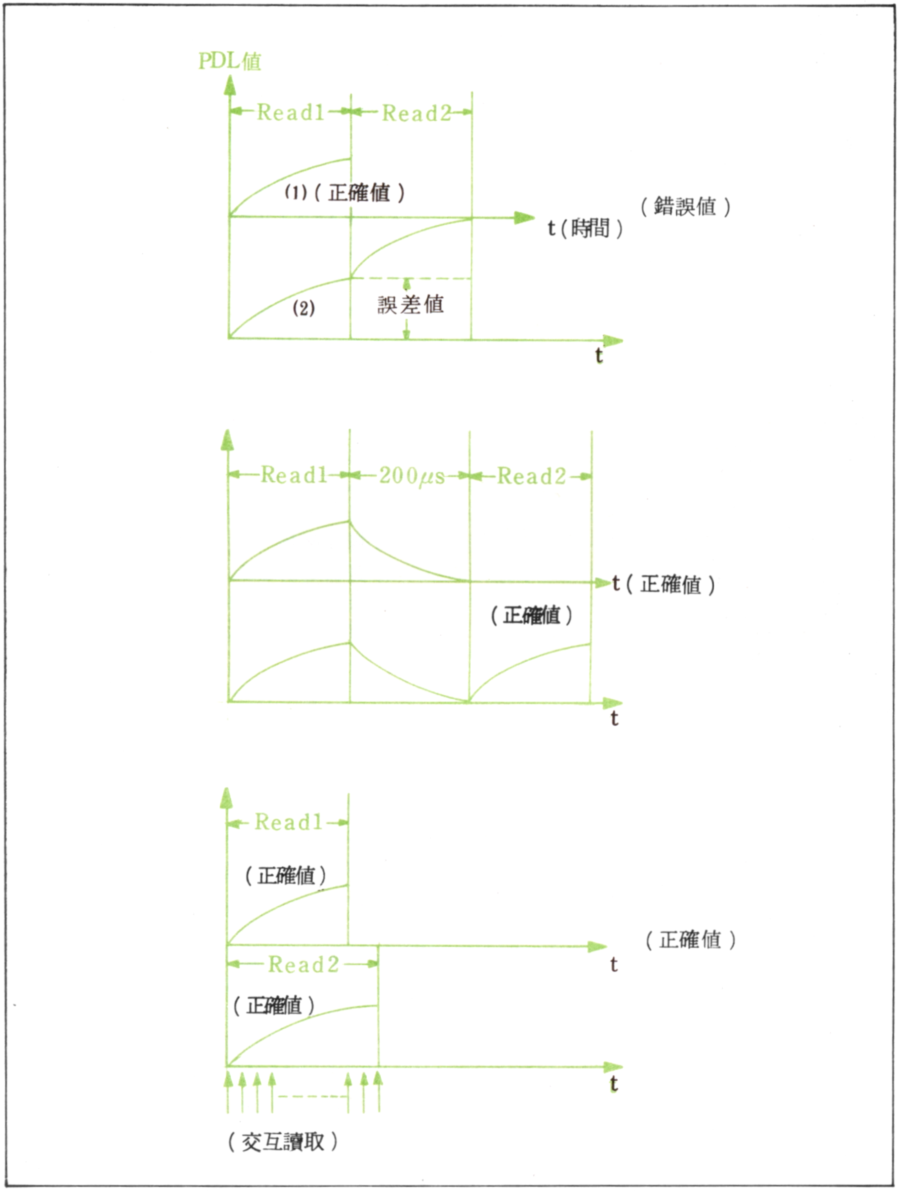


圖 3 - 6　搖桿讀取的各種情況

```
*******************************
* READ PADDLE-0 FOR X-POSITION *
*******************************
*
RPDL       LDX #$00          ; 第0號搖桿
           JSR PREAD         ; 讀PDL(0)值
SKIP       STY PDL           ; 暫存Y
           TYA
           CMP PHORIZ        ; 與目前位置比較
           BGE PADDLE3       ; 大於則到PADDLE3
           LDA PHORIZ        ; 取目前位置（小於）
           SEC
           SBC #STEP         ; 減STEP
           BGE PADDLE1       ; 大於0，則到PADDLE1
           LDA #$00          ; 小於0，則位置←0
           STA PHORIZ
           STA TPHORIZ       ; 複製新值
PADDLE1    CMP PDL           ; 是否等於PDL值？
           BGE PADDLE2       ; 大於，則到PADDLE2
           LDA PDL           ; 等於，則存原值
PADDLE2    STA PHORIZ
           JMP PADDLE6       ; OUT
PADDLE3    CMP PHORIZ        ; 大於目前值？
           BEQ PADDLE4       ; 等於，到PADDLE4
           LDA PHORIZ        ; 取目前值
           CMP #BOUNDARY     ; 超過邊界？
           BGE PADDLE4       ; 是
           LDA PHORIZ        ; 否，則加上STEP
           CLC
           ADC #STEP
           CMP PDL           ; 大於PDL值？
           BLT PADDLE5       ; 否，到PADDLE5
PADDLE4    LDA PDL
PADDLE5    STA PHORIZ
PADDLE6    STA TPHORIZ
           END
```

（參考圖3－4的流程）

程式3－4搖桿讀取程式之一（水平方向）

```
*********************************
* READ PADDLE #1 FOR Y-POSITION *
*********************************
*
START     LDX #$01
          JSR PREAD
          CPY #BOUNDARY+1
          BLT SKIPP
          LDY #BOUNDARY
SKIPP     STY PDL
          TYA
          CMP VERT
          BGE PADDLE3
          LDA VERT
          SEC
          SBC #STEP
          BGE PADDLE1
          LDA #$00
          STA VERT
          STA TVERT
PADDLE1   CMP PDL
          BGE PADDLE2
          LDA PDL
PADDLE2   STA VERT
          JMP PADDLE6
PADDLE3   CMP VERT
          BEQ PADDLE4
          LDA VERT
          CLC
          ADC #STEP
          CMP PDL
          BLT PADDLE5
PADDLE4   LDA PDL
PADDLE5   STA VERT
PADDLE6   STA TVERT
* THEN ...............
```

（說明同程式 3 - 4 ）

程式 3－5 搖桿讀取程式之二（垂直方向）

```
*************************
* DUAL PADDLE ROUTINE   *
* RETURNS VALUE OF      *
* (PDL(0),PDL(1)) FROM *
* (44,47) TO (126,127) *
*************************
                ORG  $1000
PDL0            DFS  1
PDL1            DFS  1
PTRIG           EQU  $C070
PADDLE0         EQU  $C064
PADDLE1         EQU  $C065
                LDX  #$00
                STX  PDL1
                STX  PDL0
                LDX  #$7F
*------------START TIMER
                LDA  PTRIG
*------------READ PADDLE0
LOOP            LDA  PADDLE0
                AND  #$80
                ASL
                ROL
                ADC  PDL0
                STA  PDL0
*------------READ PADDLE1
                LDA  PADDLE1
                AND  #$80
                ASL
                ROL
                ADC  PDL1
                STA  PDL1
                DEX
                BNE  LOOP
*------------END LOOP
* THEN ..........
* ...............
* ...............
```

```
                RTS
                END
```

程式 3－6 搖桿讀取程式之三（垂直及水平）

```
*************************
* EXTEND THE PADDLE RANGE *
* PDL(X)=0..370, X=0..3 ! *
*************************
*
PTRIG     EQU $C070
PADDLE0   EQU $C064
*-----------BEGIN PROG
          LDA PDLNUM
          LDA #$00         ；歸 0 並以雙位址存值
          STA COUNTER
          STA COUNTER+1
          TAY
          LDA PTRIG
          NOP
          NOP
READ      LDA PADDLE0,X
          BPL OUT
          INY
          BNE READ
          INC COUNTER+1
          LDA COUNTER+1
          CMP #$02
          BNE READ
OUT       TYA
          STA COUNTER
          RTS
COUNTER   DFS 2
PDLNUM    DFS 1
          END
```

程式 3－7 搖桿讀取程式之四（擴展 PDL 值）

程式3－4，3－5是水平及垂直方向的搖桿值讀取程式，而程式3－6是依圖3－6的說明寫成的PDL0、PDL1同時讀取的程式，這個程式的缺點是其PDL值只約在40～130之間，如果要擴展PDL值的範圍，可以採用3－7的程式。由於畫面水平座標大於255（$0 \leq X \leq 279$），而Apple的搖桿值基本上是在0～255之間，因此程式3－7在某些應用上是必要的。

3-4 嘗試寫一段動畫程式

現在，我們能做什麼呢？將程式3－0，3－1，3－2組合起來，我們可以造出一個最簡單的動畫來：

```
****************************
* SIMPLE GAME DEMO !!!     *
* USING PRO 3-0 , PRO 3-1 *
* & PRO 3-2                *
****************************
*
         ORG $0A00
*
* 0-PAGE ADDRESS
*
HIRESL    EPZ $00
HIRESH    EPZ $01
SHPL      EPZ $02
SHPH      EPZ $03
*
* TABLE ADDRESS
*
TABLO     EQU $51C0
TABHI     EQU $5100
KEY       EQU $C000
```

```
*
*==============MAIN ROUTINE
*
             STA $C050
             STA $C057
             STA $C052
*
* CLEAR SCREEN
*
CLEAR        LDA #$20
             STA HIRESH
             TAX
             LDA #$00
             STA HIRESL
             TAY
CLEAR1       STA (HIRESL),Y
             INY
             BNE CLEAR1
             INC HIRESH
             DEX
             BNE CLEAR1
*------------END OF CLEAR
SETUP        LDA #SHTAB
             STA SHPL
             LDA /SHTAB
             STA SHPH
             LDA #$12
             STA HEIGHT
             LDA #$02
             STA WIDTH
             LDA #$A0
             STA VERT
             LDA #$14
             STA HORIZ
BEGIN        JSR XDRAW
             LDA #$80
             JSR $FCA8
             JSR XDRAW
```

```
READKEY  LDA KEY
         CMP #$95
         BEQ RIGHT
         CMP #$88
         BEQ LEFT
         CMP #$C1
         BEQ UP
         CMP #$DA
         BEQ DOWN
         JMP BEGIN
*==================END OF MAIN ***
LEFT     LDA HORIZ
         CMP #$00
         BEQ BEGIN
         DEC HORIZ
         JMP BEGIN
RIGHT    LDA HORIZ
         CMP #$25
         BEQ BEGIN
         INC HORIZ
         JMP BEGIN
UP       LDA VERT
         CMP #$00
         BEQ BEGIN
         DEC VERT
         JMP BEGIN
DOWN     LDA VERT
         CMP #$AA
         BEQ BEGIN
         INC VERT
         JMP BEGIN
*------------END OF KEY ROUTINE
XDRAW    LDY VERT
         STY VERT1
XDRAW1   LDY VERT1
         JSR GETADR
         LDY #$00
         LDX #$00
```

```
XDRAW2   LDA (SHPL,X)
         EOR (HIRESL),Y
         STA (HIRESL),Y
         INC SHPL
         INY
         DEC WIDTH1
         BNE XDRAW2
         LDA WIDTH
         STA WIDTH1
         INC VERT1
         DEC HEIGHT
         BNE XDRAW1
         LDA #$12
         STA HEIGHT
         LDA #SHTAB
         STA SHPL
         RTS
*------------END OF XDRAW
*
* GETADR ROUTINE
*
GETADR   LDA TABLO,Y
         CLC
         ADC HORIZ
         STA HIRESL
         LDA TABHI,Y
         STA HIRESH
         RTS
*===============END OF PROGRAM
WIDTH1   DFS 1
WIDTH    DFS 1
HEIGHT   DFS 1
VERT     DFS 1
VERT1    DFS 1
HORIZ    DFS 1
SHTAB    HEX 4000400040006001
         HEX 7003700378077807
         HEX 7C0F7D2F67396739
```

```
HEX 67397F3F6F3D4738
HEX 03300120
END
]
```

程式 3-8 火箭程式

```
0A00- 8D 50 C0 8D 57 C0 8D 52
0A08- C0 A9 20 85 01 AA A9 00
0A10- 85 00 A8 91 00 C8 D0 FB
0A18- E6 01 CA D0 F6 A9 D9 85
0A20- 02 A9 0A 85 03 A9 12 8D
0A28- D5 0A A9 02 8D D4 0A A9
0A30- A0 8D D6 0A A9 14 8D D8
0A38- 0A 20 8E 0A A9 80 20 A8
0A40- FC 20 8E 0A AD 00 C0 C9
0A48- 95 F0 1C C9 88 F0 0B C9
0A50- C1 F0 21 C9 DA F0 2A 4C
0A58- 39 0A AD D8 0A C9 00 F0
0A60- D8 CE D8 0A 4C 39 0A AD
0A68- D8 0A C9 25 F0 CB EE D8
0A70- 0A 4C 39 0A AD D6 0A C9
0A78- 00 F0 BE CE D6 0A 4C 39
0A80- 0A AD D6 0A C9 AA F0 B1
0A88- EE D6 0A 4C 39 0A AC D6
0A90- 0A 8C D7 0A AC D7 0A 20
0A98- C4 0A A0 00 A2 00 A1 02
0AA0- 51 00 91 00 E6 02 C8 CE
0AA8- D3 0A D0 F2 AD D4 0A 8D
0AB0- D3 0A EE D7 0A CE D5 0A
0AB8- D0 DA A9 12 8D D5 0A A9
0AC0- D9 85 02 60 B9 C0 51 18
0AC8- 6D D8 0A 85 00 B9 00 51
0AD0- 85 01 60 02 02 12 00 12
0AD8- 11 40 00 40 00 40 00 60
0AE0- 01 70 03 70 03 78 07 78
```

```
0AE8- 07 7C 0F 7D 2F 67 39 67
0AF0- 39 67 39 7F 3F 6F 3D 47
0AF8- 38 03 30 01 20
*
```

程式 3 - 8 的機械碼（請用BLOAD取入畫面位址表）

程式3－8基本上是由程式3－0，3－1，3－2組合起來的，由119～125行是GETADR，92～114是XDRAW，60～90行是READKEY，而31～42行是程式0－5的底色程式，只不過現在畫的是黑色的底色，所以較為簡短。程式執行時圖3－3的圖形表已存到由SHTAB＝$0AD9開始的36個位址內。本程式是使用LISA寫成的，所以#SHTAB代表SHTAB的低位元組（$D9），／SHTAB代表高位元組（$0A）。圖形長寬的值在48～51行設定成HEIGHT＝#$12，WIDTH＝#$02。與程式3－1不同的是我們在XDRAW當中，以WIDTH1來代替WIDTH及VERT1代替VERT來做運算，這是因為XDRAW會改變VERT及LNGH的值，因此必須加以保留下次使用。同樣的，HEIGHT及SHPL的值也在110～113行取回原值。

這個程式有些必要檢討的地方，如果你按下′A′鍵圖形會徐徐上移，而按″→″時，向右移動却很不自然，這是因為上移時是改變VERT的值，因此每次上升一線（即垂直一點的間隔），而水平移動却是一次移動了一個位元組（水平的7點），這便是我們在第0章所強調的一個問題，在第4章我們將學到解決這個問題的方法。另外，我們在57～58行使用了$FCA8的ROM延時副程式，其目的是為了使畫出的圖形可以暫存於螢幕上而不要馬上消去。我們只是暫時的借用這個副程式，以後程式會愈來愈龐大，這時$FCA8所佔用的時間便可用來執行其它工作了。請你也嘗試以附錄A的飛機（圖A－3）來取代程式3－8的火箭，一方面可以熟悉程式的修改，另外也可以觀察一下彩色圖形的情形。

如果將圖 1 — 1 當成背景，則結果又如何呢？您會發現，背景不會因爲火箭經過而受破壞，但是經過不同底色的背景，白色的火箭却改變了顏色，同時出現了很嚴重的斑點現象。這便證實了我們所說EOR畫圖理論了。可是我們曾提過許多解決的方法，現在便來看看，如何在複雜的背景上畫圖的方法………。（要將圖 1 — 1 的背景移到程式 3 — 8 來使用，只要將圖 1 — 1 畫出來，RESET 後執行程式 3 — 8 便可以了，不過 3 — 8 的畫面消除副程式 CLEAR 必須以JMP指令跳過去，背景才不會被清除而消失掉）。

3 5 火箭與背景的問題

所以，您發現背景確是一個大問題。爲了同時保存背景圖案又要使所畫的圖形完美，我們在 1 — 4 已討論了許多對策，現在就來看看這些方法如何實際的應用在程式裡頭：

```
**************************
* SIMPLE GAME DEMO !!!   *
* USING PRO 3-0 , PRO 3-1 *
* & PRO 3-2              *
* IN COLOR BACKGROUND    *
**************************
*
         ORG $0A00
*
* DEFINE STORAGE
*
LNGH1    DFS 1
LNGH     DFS 1
DEPTH    DFS 1
VERT     DFS 1
```

```
VERT1     DFS 1
HORIZ     DFS 1
*
* 0-PAGE ADDRESS
*
HIRESL    EPZ $00
HIRESH    EPZ $01
SHPL      EPZ $02
SHPH      EPZ $03
BAKGRL    EPZ $04
BAKGRH    EPZ $05
*
* TABLE ADDRESS
*
YVERTL    EQU $1000
YVERTH    EQU $10C0
SHADR     EQU $1180
KEY       EQU $C000
*
*=============MAIN ROUTINE
*
          STA $C050
          STA $C057
*
* CLEAR SCREEN
*
CLEAR     LDA #$20
          STA HIRESH
          TAX
          LDA #$00
          STA HIRESL
          TAY
          LDA #$AA
CLEAR1    STA (HIRESL),Y
          INY
          BNE CLEAR1
          INC HIRESH
          DEX
```

```
        BNE  CLEAR1
*------------END OF CLEAR
SETUP   LDA  #SHADR
        STA  SHPL
        LDA  /SHADR
        STA  SHPH
        LDA  #BAKADR
        STA  BAKGRL
        LDA  /BAKADR
        STA  BAKGRH
        LDA  #$12
        STA  DEPTH
        STA  HORIZ
        LDA  #$02
        STA  LNGH
        LDA  #$A0
        STA  VERT
BEGIN   JSR  DRAW
        LDA  #$80
        JSR  $FCA8
        JSR  BAKDRAW
READKEY LDA  KEY
        CMP  #$95
        BEQ  RIGHT
        CMP  #$88
        BEQ  LEFT
        CMP  #$C1
        BEQ  UP
        CMP  #$DA
        BEQ  DOWN
        JMP  BEGIN
*==================END OF MAIN ***
LEFT    LDA  HORIZ
        CMP  #$00
        BEQ  BEGIN
        DEC  HORIZ
        JMP  BEGIN
RIGHT   LDA  HORIZ
```

```
              CMP #$26
              CMP #$25
              BEQ BEGIN
              INC HORIZ
              JMP BEGIN
UP            LDA VERT
              CMP #$00
              BEQ BEGIN
              DEC VERT
              JMP BEGIN
DOWN          LDA VERT
              CMP #$B3
              BEQ BEGIN
              INC VERT
              JMP BEGIN
*-------------END OF KEY ROUTINE
DRAW          LDY VERT
              STY VERT1
DRAW1         LDY VERT1
              JSR GETADR
              LDY #$00
              LDX #$00
DRAW2         LDA (HIRESL),Y
              STA (BAKGRL,X)
              LDA (SHPL,X)
              LSR
              ORA (SHPL,X)
              EOR #$FF
              AND (HIRESL),Y
              ORA (SHPL,X)
              STA (HIRESL),Y
              INC BAKGRL
              INC SHPL
              INY
              DEC LNGH1
              BNE DRAW2
              LDA LNGH
              STA LNGH1
```

```
                INC  VERT1
                DEC  DEPTH
                BNE  DRAW1
*-----------BEGIN RESTORE
                LDA  #$12
                STA  DEPTH
                LDA  #BAKADR
                STA  BAKGRL
                LDA  #SHADR
                STA  SHPL
                RTS
*------------END OF  DRAW
BAKDRAW         LDY  VERT
                STY  VERT1
                LDA  LNGH
                STA  LNGH1
BAKDRAW1        LDY  VERT1
                JSR  GETADR
                LDY  #$00
                LDX  #$00
BAKDRAW2        LDA  (BAKGRL,X)
                STA  (HIRESL),Y
                INC  BAKGRL
                INY
                DEC  LNGH1
                BNE  BAKDRAW2
                LDA  LNGH
                STA  LNGH1
                INC  VERT1
                DEC  DEPTH
                BNE  BAKDRAW1
*-----------BEGIN RESTORE
                LDA  #$12
                STA  DEPTH
                LDA  #BAKADR
                STA  BAKGRL
                RTS
*-----------END OF BAKDRAW
```

```
169 *
170 * GETADR ROUTINE
171 *
172 GETADR    LDA YVERTL,Y
173           CLC
174           ADC HORIZ
175           STA HIRESL
176           LDA YVERTH,Y
177           STA HIRESH
178           RTS
179 BAKADR    DFS 36
180 *===================END OF PROGRAM
181           END
```

程式 3 - 9　有彩色背景的火箭

這個程式就像我們在 2 － 3 節所說的，以AND，ORA ，EOR 等運算來造出一個與底色互不干擾的畫圖動作。基本上這是程式 3 － 8 的推演，所以其機械碼就不再重複列出，程式中需要說明的首先是第118 行的LSR，這是爲了將 117行所取到的圖形表移位，然後再與圖形表做 O R 運算，如此便可以將一個彩色的實心圖形變成白色圖形，因爲我們由 2 － 3 節知道必須以白色的圖形參與運算才行，可是並不是每種圖形都可以這麼簡單的以 LSR便可以造出白色圖形來，譬如圖形中間有黑色空洞時便會發生背景由空洞內出現，有時候這正符合您的需要，但如果您要消除這種情況的話，就必須造出另外一個白色圖形表來 EOR ＃＄FF才行。

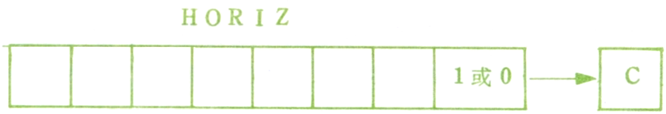
當然您會發現圖形的邊緣仍舊有黑色的斑點，這是因爲LSR 的緣故，使得原本爲奇數點的線段變成偶數（多出一點），邊緣就出現黑色輪廓了，如果將 118行的LSR換成NOP，則火箭尖端的黑點就不見了，但圖形却因而受到損壞。如果我們另外造出第 2 組圖形表，將圖形上每一條水平線中間的 0 （空洞）均換成 1 ，然後將 117 ～ 119 行的指令換成LDA（SHPL

2 ，X ），就能造出完美的圖形來了，SHPL 2 是第 2 組圖形表的起始位址。這種方式的唯一缺點是圖形必需不能向內凹，譬如例子裡所用的火箭，其兩個窗戶及尾巴缺口都變成黑色的了。請依實況來選用適當的方法！

3-6 彩色圖形的水平移動——奇偶圖形表示法

如果將圖 A－3 的飛機以程式 3－8 畫出在螢幕上運動，結果其顏色會因所在的位元組而改變，這個問題我們在附錄 A 已經提過了。在第 4 章的預移圖序法當中，我們將有很方便的解決方法，可是目前的XDRAW 方式是每次移動一個水平的位元組，如此奇、偶位元組的位置變換，顏色也就改變了。當然我們可以一次移動兩個位元組以保持位元組的奇偶次序，可是在程式 3－8 我們已看過因為移動間隔太大而使動作看來不自然，所以一次移動兩個位元組並不是很好的方法。

一個解決的方法是造出兩個圖形表，當圖形最左端位置為奇數位元組時，畫出對應的奇數圖形表，而圖形最左端位置為偶數時，則畫偶數圖形表，這樣顏色便足以保持了。那麼怎樣才能在每個位置上皆選到正確的圖形表呢？




```
LDA  HORIZ        ；取水平座標值（ 0～39）
LSR               ；取 b0 ，若奇數則 b0 ＝1
BCS  ODD          ； b0 ＝1 則取奇數圖形表
```

圖 3－7　奇偶位置的測試

因爲一個位元組的每一位元除了b 0之外，所有位元均爲2的整數次方，只要將 HORIZ 的最低位元移出來測試其值（移到進位旗號C），若最低位元爲0，則 HORIZ 爲偶數，反之則爲奇數。

```
*****************************
* DOUBLE SHAPE TABLE METHODE *
*****************************
*       ORG $XXXX
HORIZ     DFS 1
SHPL      EPZ $02
SHPH      EPZ $03
*
* GET TABLE ROUTINE
*
          CLC
          LDA HORIZ
          LSR
          BCS ODD
EVEN      LDA #SHADR
          STA SHPL
          LDA /SHADR
          STA SHPH
          JMP CONT
ODD       LDA #SHADR+36
          STA SHPL
          LDA /SHADR+36
          STA SHPH
CONT      RTS
```

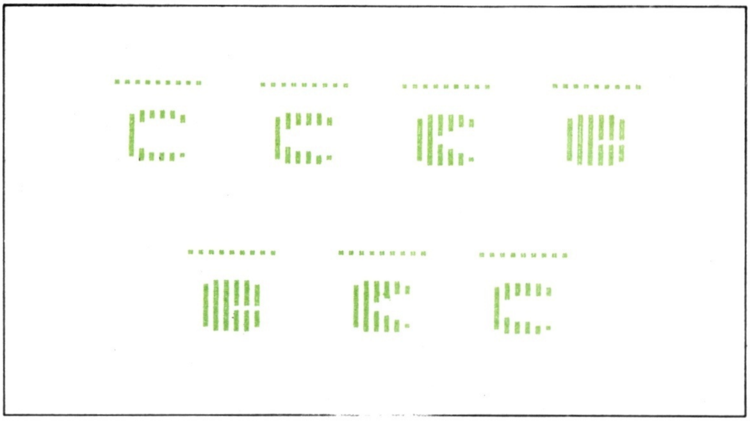
程式 3-10　雙圖形表的使用法

本章提要：

上一章的火箭水平移動並不是很自然，而垂直移動就平滑多了，這是水平移動距離太大（距離一位元組）的原因，如果要縮小移動距離呢？位元間的移動，當然要使用預移圖序法了！在本章當中：

1. 我們將以實例來說明預移圖序的使用法
2. 我們將探討彩色圖序的解決方法
3. 除了畫面垂直位址查表之外，更進一步的做水平位址查表
4. 另外，我們也將嘗試第 4 種繪圖（消圖）方式——不用EOR、AND、ORA、………的直接繪圖法

4 走吧！大嘴巴！



4-0 預移圖序法的徹底研究

所以，我們便來畫一組圖序吧！除了在第二章所提到的各項預移圖序法的優點之外，同時我們也想到，既然預移圖序的原理是類如卡通動畫，如果除了位置的移動之外也改變其形狀或姿勢，這樣看起來豈不是與卡通人物一樣效果了呢？所以，除了第 2 章的圖 2 － 3 或 2 － 6 這種固定形狀

姿態的圖序之外，我們也可以畫出動態的圖序來：

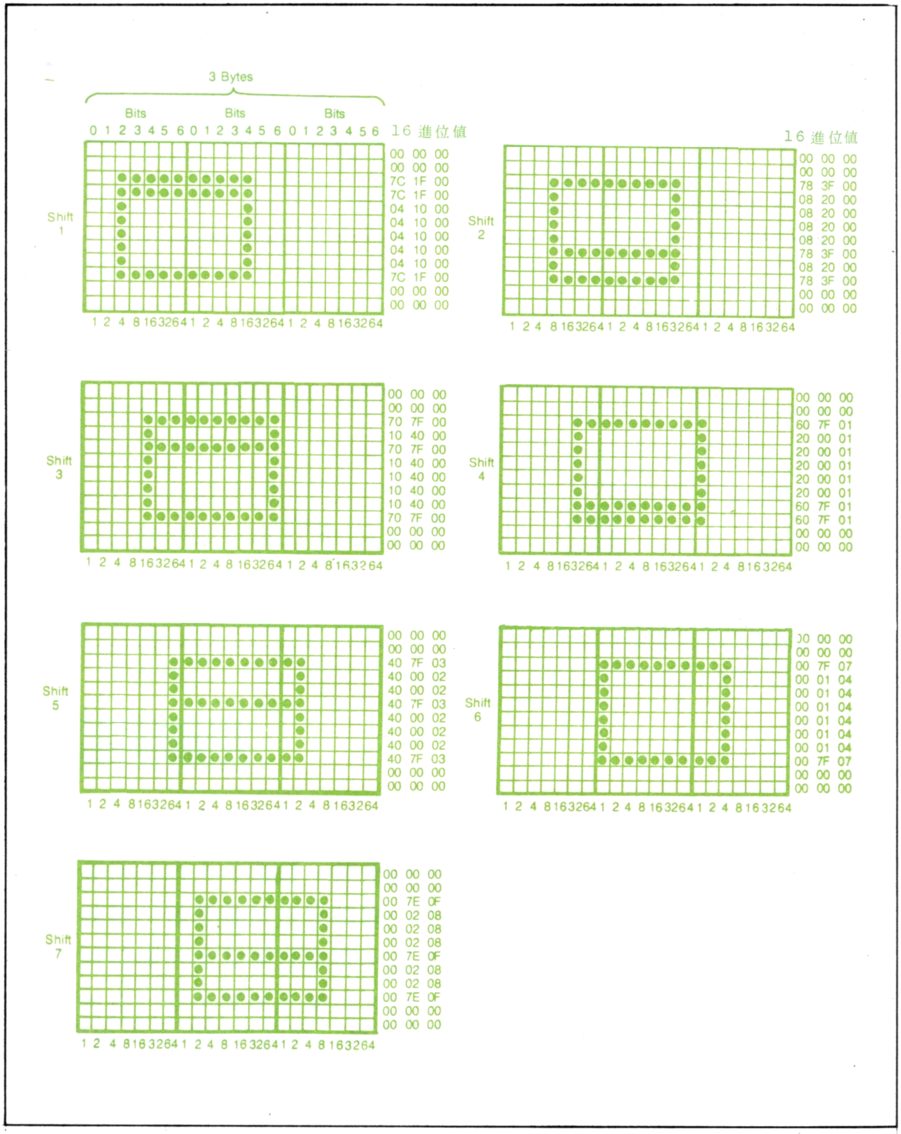


圖 4 — 0　動態的預移圖序

```
5400- 00 10 00 34 00 58 00 7C
5408- 00 A0 00 C4 00 E8 03 0C
5410- 00 00 00 00 00 00 7C 1F
5418- 00 7C 1F 00 04 10 00 04
5420- 10 00 04 10 00 04 10 00
5428- 04 10 00 7C 1F 00 00 00
5430- 00 00 00 00 00 00 00 00
5438- 00 00 78 3F 00 08 20 00
5440- 08 20 00 08 20 00 08 20
5448- 00 78 3F 00 08 20 00 78
5450- 3F 00 00 00 00 00 00 00
5458- 00 00 00 00 00 00 70 7F
5460- 00 10 40 00 70 7F 00 10
5468- 40 00 10 40 00 10 40 00
5470- 10 40 00 70 7F 00 00 00
5478- 00 00 00 00 00 00 00 00
5480- 00 00 60 7F 01 20 00 01
5488- 20 00 01 20 00 01 20 00
5490- 01 20 00 01 60 7F 01 60
5498- 7F 01 00 00 00 00 00 00
54A0- 00 00 00 00 00 00 40 7F
54A8- 03 40 00 02 40 00 02 40
54B0- 7F 03 40 00 02 40 00 02
54B8- 40 00 02 40 7F 03 00 00
54C0- 00 00 00 00 00 00 00 00
54C8- 00 00 00 7F 07 00 01 04
54D0- 00 01 04 00 01 04 00 01
54D8- 04 00 01 04 00 01 04 00
54E0- 7F 07 00 00 00 00 00 00
54E8- 00 00 00 00 00 00 00 7E
54F0- 0F 00 02 08 00 02 08 00
54F8- 02 08 00 7E 0F 00 02 08
5500- 00 02 08 00 7E 0F 00 00
5508- 00 00 00 00
```

圖 4－0 的圖序表

當然圖 4 － 0 是為了易於解說及學習，因此選擇簡單而效果明顯的圖形，所以，請您自行發展出更多更有趣的圖序來吧！至於圖 4 － 0 的圖序

表當中，最前頭的16個位元組並非圖形的資料，其中由第 0 ～第13位元組分別存放了 7 個圖序的位置索引，亦即起始位址的高低位元組（ 7 個圖序 × 2 位元組 = 14位元組），而第14及15位組則是存放圖形的寬度及高度。所以眞正的圖形資料是由＄5410開始的，也就是說第 0 號圖序資料的起始位址是＄5410 ，可是由＄5400～＄5413 不就是用來存於起始位址的位元組嗎？那麼＄5400 、＄5401 的值應該是第 0 號圖序的起啓位址（＄54，＄10）才對，但事實上該二位址的值却是（＄00，＄10），原來此處所存的是相對於整個圖序表的位址，而不是絕對位址（眞正對應於位址線的位址），也就是說，由＄5400～＄540D 所存的是每個圖序相對於圖序表開頭（＄5400 ）的距離。既然是距離，就必須加上＄5400 才是眞正的位址，而使用6502 的索引定址法便可以達到這個目的。問題是，為什麼不直接寫出絕對位址，而要用相對位址呢？因為，圖形種類一旦增多，圖序表變得很龐大時，圖序表開頭位址一旦改變，則勢必每個圖形的每個圖序的位置索引，此時若使用的是相對位址方式，則只要改變整個圖序的開頭位址便可以了，因為相對距離並未改變。所以，當您創造一組圖序之後，可以附加到任何程式當中，圖序本身不需任何修改。

雙位元移位法

在 2 － 2 節我們提到關於彩色圖形的移位問題，我們是以間隔兩個位元的方式來移位，所以 7 個圖序共移了12個位元而佔了14個位元的空間，如果圖形寬度是 n 個位元組的話，則圖序寬度是 n ＋ 2 個位元組。但是圖 4 － 0 的圖形寬度佔用了 2 個位元組，但圖序寬度却只用了 3 個位元組，而且仔細觀察圖中日字形方格內的水平線，好像也不依次序排列，到底這個圖序表有什麼特殊之處呢？

請依 1 、 3 、 5 、 7 、 2 、 4 、 6 的次序來觀察圖 4 － 0 的 7 個圖序吧！因為圖形是否移動兩個位元，除了由圖序本身來加以設定之外，也可以由程式中每次水平位置加 2 來畫圖，結果是一樣的，問題是在於我們的

程式能否自動的控制顏色而不受移位的影響。現在回頭來看圖 4 － 0 的圖序，如果我們以 1、2、3、4、5、6、7 的順序來存放圖序表，但以 1、3、5、7、2、4、6 的順序來畫圖形的話，其結果是與圖 2 － 6 的效果一樣的，也就是說這時圖形的顏色不受影響。這是因爲眞正使圖形移動的是垂直與水平座標這兩個變數，而不是圖序本身，因此當第 7 號圖序畫完之後再畫第 2 號圖序不一定會令圖形往回頭跑，這時只要令水平位元組加一，則第 2 號圖序恰好超前第 7 號圖序兩個位元組（向右兩個位元組），這種效果與圖 2 － 6 是完全相同的：圖形每次水平移動兩位元，而顏色不受影響。採用這種方式，不但可以節省一個位元組的空間，畫圖速度也得以加快（本例中，每個圖序少劃了12 個位元組）。

所以，不但圖序移動的次序無誤，顏色也保持不變。但是，您還是發現了不對勁的地方：如果我們只是依上述的原則，畫出間格一個位元的圖序，然後間隔一個號碼來取用圖序值，對於固定形狀的圖序來說是沒什麼問題的，但如果像圖 4 － 0 這種動態的圖序呢？隨著位置移動兩個位元，動作是否也要間格一個圖序呢？如此，會產生生硬不自然或動作太快的結果，就像是卡通或電影的膠捲底片被跳著來放映一般（去掉偶數或奇數的畫面）。由於 Apple 畫面的構造，令我們不得不使用雙位元移位法，但是圖序的動作却是不必要也跟著做間隔移動的。因此，畫圖序時，先將′□′字形的方格畫好，然後再將′一′以 1、3、5、7、2、4、6 的次序塡入到方格內，這便是圖 4 － 0 的由來了。所以，我們便以最有效的′雙位元移位法′造出了′動態預移圖序表′！

直接繪圖法

除上述的優點之外，圖 4 － 0 還具備了一項快速繪圖的功能。我們在第 3 章已經使用了EOR、ORA與AND的方式來繪圖，並且體會了其繪圖、消圖及底色處理的能力。但是 2 － 3 節我們另外還提到了一種稱爲′直接繪圖法′的方式，使用這種方式的圖形，除了本身的圖形之外，其四週圍

還要以底色來環繞，每次一個新圖形畫出來的時候，舊的圖形便被蓋住了，但是新舊圖形之間總有其移位的差距而不能以新圖形來遮蓋吧？這便是圖形周圍環繞著底色的功能了，只要移位差距不超過圍繞的底色寬度，則舊圖形雖然不被新圖形蓋，却被周圍的底色蓋住了。這便是最迅速的直接畫圖方式。

在圖 4 － 0 當中，我們假設畫面是黑色的底色，因此圖 4 － 0 的每一個圖序上下均保持了兩條黑色水平線的輪廓，而左右亦分別至少保持了 2 及 3 個黑色位元的寬度，因此，圖 4 － 0 可以看成在一個21點 × 12線的方塊內畫出10點 × 8 線的圖序，由於我們的雙位元移位法使得 7 個圖序只間隔了 6 個位元，因此左右便有 2 及 3 個位元的空間。垂直方面，則因爲圖序並不向上下移動，所以永遠保有 2 個水平線的距離。

那麼，只要圖形移動距離不超過 2 個位元，舊圖形永遠是會被新圖形遮蓋住的，這種方式雖然在非單一底色的場合有其不便之處（可以暫存底色及改變輪廓顏色），但却不失爲最快速有效的繪圖方式！

4 1 預移圖序的繪圖程序

程式 4 － 0 便是用來畫出預移圖形的副程式，只要由主程式設定垂直及水平座標，然後呼叫程式 4 － 0 便可以畫出一個圖形來。

程式當中，XBYTE 是存放＇水平座標值所對應＇的位址，而 XBIT 則是對應的位元，譬如水平座標爲73這點對應於水平線上第10位元組的第 3 位元，因此（ XBYTE ）＝＄0 A 而（ XBIT ）＝＄03。VERT 及 SCX則分別爲垂直及水平座標變數，主程式的垂直及水平座標值便是輸入到這兩個變數位址內。此處 VERT是與第 3 章程式中的 VERT 相同，但 SCX却與第 3 章當中的 HORIZ 不同，因爲 HORIZ 是存放水平線上的位址（ 0 ～39），等於是此處的XBYTE ，而 SCX却是水平線上的座標值（ 0 ～ 279），這也便是我們的主題之一：如何將程式 3 － 8 那種不自然的

水平移位動作，改進成更自然而順暢！（因為位置解析度已由 0 ～39改進至 0 ～ 279 了）

程式由第30行開始執行，31 行將水平位置取入 A ，36行至44行將 A 的內含值除以 7 ，其商存入 XBYTE ，餘數存入 XBIT，因此便得到對應的水平位址及位元。48行至 53行由圖序表中取得圖形的寬度及高度。

由 57行至63行是一個常用的雙位元組查表方式，首先57行取得XBIT的值，用以判斷應該取用第幾號圖序值，圖序值是存放於圖 4 － 0 的圖序表，由於圖序起始位址共要兩個位元組來表示，因此要查第 3 號圖序的起始位址，便要取第 6 及第 7 位元組的資料（以圖 4 － 0 的圖序表起始位址 $ 5400 為第 0 位址開始算），也就是 3 × 2 及 3 × 2 ＋ 1 的位置，因此若要取第 n 號圖序的起始位址，便要取第 n × 2 及 n × 2 ＋ 1 位置的內含值，所以在57行取得 XBIT之後便ASL（乘 2 ）取得 SHTAB ＋ Y ＝ SHTAB ＋ 2n 位置的內含值然後存入X ，INY之後便取得 SHTAB＋2n ＋ 1 位置的內含值。

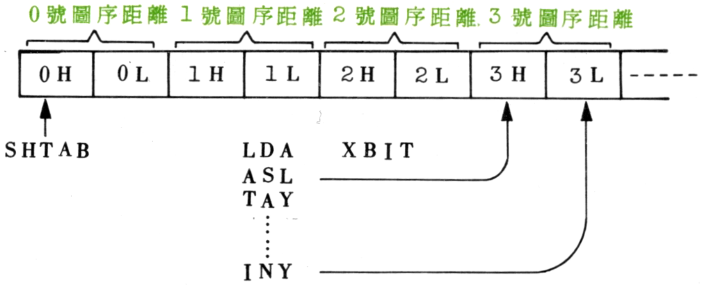


圖 4 - 1　每號圖序位址至圖序表開頭的距離

```
!L
 ;
*************************
* DRAWING PRE-SHIFT GRAPH *
* USING  LOOK-UP TABLE !! *
*************************
*
          ORG $A00
*
* 0-PAGE ADDRESS
*
XBYTE     EPZ $00
XBIT      EPZ $01
VERT      EPZ $02
XCOUNT    EPZ $03
SCX       EPZ $04
HIRESL    EPZ $06
HIRESH    EPZ $07
WIDTH     EPZ $08
HEIGHT    EPZ $09
*
* NON-0-PAGE ADDRESS
*
TABHI     EQU $5100
TABLO     EQU $51C0
SHTAB     EQU $5400
*******************************
* MAIN ROUTINE BEGIN HEHE !!! *
*******************************
*
START     LDY #$00
          LDA SCX          ;取水平座標
          CLC
*
* DIVIDE BY '7' ROUTINE
*
DLOOP     CMP #$07         ;若小於 7 就不除 7
          BCC DDONE
          SEC
```

```
          SBC #$07          ; 減 7
          INY               ; Y＝Y＋1
          CLC
          JMP DLOOP         ; 繼續減 7
DDONE     STA XBIT          ; 餘數＝XBIT ← A
          STY XBYTE         ; 商 XBYTE ←Y
*
* SET UP WIDTH AND HEIGHT
*
          LDY #$0E          ; ( $ 540E ) ＝寬度值
          LDA SHTAB,Y
          STA WIDTH
          INY               ; ( $ 540F ) ＝高度值
          LDA SHTAB,Y
          STA HEIGHT
*
* GET SHAPE TABLE VALUE
*
          LDA XBIT          ; 取位元值＝圖序號碼
          ASL               ; 乘 2
          TAY
          LDA SHTAB,Y       ; 取到對應的圖序起
          TAX               ; 始位址的距離並
          INY               ; 存於 A 及 X
          LDA SHTAB,Y
*
* AUTO-MODIFY P.S.#
*
          CLC               ; 加上圖序表起始
          ADC #SHTAB        ; 位址 ( $ 5400 ) 然
          STA BLOCK1+1      ; 後存到 BLOCK 1
          TXA               ; 之後的兩個位置內
          ADC /SHTAB
          STA BLOCK1+2
*
* BEIGN DRAWING ONE SHAPE
*
          LDX #$00
```

```
LOOP1     LDA WIDTH          ;設定寬度(水平)
          STA XCOUNT
          LDY VERT           ;取畫面垂直位址
          CPY #$C0           ;超過底線?
          BCC YTABLE
          LDY #$29           ;若超過底線則
          BCS LOOP2          ;以LOOP2 的CPY
*                            ;來限制畫圖
* LOOKING UP SCREEN ADDR TABLE
*
YTABLE    LDA TABLO,Y        ;取該線的起始
          STA HIRESL         ;位址並存入
          LDA TABHI,Y        ;(HIRESL,H )
          STA HIRESH
          LDY XBYTE
*
* DRAW A LINE
*
LOOP2     CPY #$28           ;超過右邊界線
          BCS CONT           ;則不畫圖
BLOCK1    LDA SHTAB,X        ;取圖序表值
          STA (HIRESL),Y     ;畫圖形
CONT      INX
          INY
          DEC XCOUNT         ;向右一位元組
          BNE LOOP2
*
* END OF DRAWING ONE LINE
*
          INC VERT           ;向下一線
          DEC HEIGHT         ;計數減1
          BNE LOOP1
*
* END OF DRAWING ONE SHAPE
*
          RTS
          END
```

程式4—0

由76行至108行則是畫圖的廻圈，其中87行至91行是畫面位址查表，垂直位置（水平掃描線號碼）由 VERT取入Y，然後查表存入（HIRESL，H），接著將水平位址 XBYTE 取入Y便開始95行至102行的畫水平線的廻圈了。由於77行將圖形寬度取入 XCOUNT，因此97行將圖序值取入到A，98行將圖形資料畫到螢幕上，如此一直到 XCOUNT= 0，這樣剛好畫完圖形寬度的一條水平線，而外面的廻圈 LOOP 1，則令LOOP 2 重複到畫完整個圖形高度。

```
0A00- A0 00 A5 04 18 C9 07 90
0A08- 08 38 E9 07 C8 18 4C 05
0A10- 0A 85 01 84 00 A0 0E B9
0A18- 00 54 85 08 C8 B9 00 54
0A20- 85 09 A5 01 0A A8 B9 00
0A28- 54 AA C8 B9 00 54 18 69
0A30- 00 8D 5B 0A 8A 69 54 8D
0A38- 5C 0A A2 00 A5 08 85 03
0A40- A4 02 C0 C0 90 04 A0 29
0A48- B0 0C B9 C0 51 85 06 B9
0A50- 00 51 85 07 A4 00 C0 28
0A58- B0 05 BD A0 54 91 06 E8
0A60- C8 C6 03 D0 F1 E6 02 C6
0A68- 09 D0 D1 60
```

程式4－0　的機械碼表

自動位址設定法

或許您已發現到，我們跳過程式 4－0 當中的67行至72行沒有加以說明。這 6 行程式也是一種常用的技巧，它將由57行63行取到的圖序距離加

上圖序表的開頭位址之後，存入到BLOCK1之後的兩個位址內，而BLOCK 1便是97行的LDA SHTAB，X，因此表面上97行是以SHTAB 為基準位址，然後做X索引，其實這個基準位置並非是 SHTAB ，而是由67行至72行所存進來的值，也就是說這時程式自動修改了程式本身的內容！每次所取圖序不同時，便將97行後兩位址的值修改成對應的圖序位址，然後再以X索引取出眞正所要的資料。

這種方式以直接位址的設定，來取代取址手續，一方面增快了執行速度，另外也節省了幾個指令空間。在程式1－2我們也曾遇見類似的情況。

程式4－0的機械碼表共只有$6C個位元組，請鍵入這個程式並保存起來。程式4－1便是一個簡單的BASIC程式，用以說4－0副程式的用法。程式4－1的特殊之處在於70行及80行的公式，其中（XP＜80）及（XP＞160）是邏輯運算其值等於0或1，因此70行的公式其結果只有X＝X、X－2、X＋2、3種，所以圖序才能以1、3、5、7、2、4、6的方式間隔畫圖。另外也由於X每次只前進或後退兩格，所以使得前述的直接畫圖法得以實現，否則如果X值變化太大，新圖形與舊圖形的差距拉大，舊圖形就無法以新圖形來掩蓋了，請嘗試將70行及80行分別改成X＝XP及Y＝YP，結果如何呢？您會發現畫面上留下許多圖形無法消除了。這也說明了我們採用圖3－4的搖桿讀取方式的原因之一了。

```
0  REM ******************
1  REM * PRE-SHIFT DEMON *
2  REM ******************
3  REM
10 X = 128:Y = 128: HGR
20  POKE  - 16302,0
25  REM * SET VIEW PORT *
30  HCOLOR= 3: HPLOT 0,0 TO 255,0
      TO 255,191 TO 0,191 TO 0,0
```

```
40  REM * DRAWING PICTURE !!! *
50  POKE 2,Y: POKE 4,X: CALL 2560

60 XP =  PDL (0):YP =  PDL (1)
65  REM * SWITCH PADDLE VALUE *
70 X = X - 2 * (XP < 80) + 2 * (X
     P > 160)
80 Y = Y - 2 * (YP < 80) + 2 * (Y
     P > 160)
90  GOTO 50
```

程式4－1　動態圖序示範

由於 POKE I ，J 的J 值必須小於255 ，因此程式4－1執行時，必須控制搖桿使圖形不要碰到邊線（小於255 ，大於0 ），否則程式便停止了。執行程式4－1時，請不要忘了鍵入①程式4－0的機械碼②圖4－0的圖序表以及③以程式0－2來造出畫面位址表。至於彩色移位的情況，請見4－3節的程式，便可證明我們的方法對彩色圖形亦能處理。

4-2 程式4-0與3-8的比較

程式4－0本身是一個副程式，其作用等於是程式3－8的XDRAW副程式（99行至121行），只不過其GETADR是包含於程式4－0內，而3－8的XDRAW只是叫用GETADR而不包含GETADR。至於VERT、HIRESL、HIRESH、DEPTH、LENGTH（即HEIGHT與WIDTH）等變數在程式4－0與3－8均具同樣的意義。其中最大的差別在於3－8只有HORIZ（0～39）這個變數，而4－0是將SCX（0～255）細分成XBYTE（等於是HORIZ）及XBIT，所以其水平移位的動作就更細緻順暢了。

至於眞正畫出一個圖形時，二者的方式均是依照圖 3 — 0 的方法來完成的。不過程式 3-8 是以（ SHPL，H ）這組指位器來指出圖形表位址（53 行～56 行），而以（ HIRESL，H ）指位器來指出畫面位址，所以整個動作就是將（ SHPL ，H ）所指的資料，搬到（ HIRESL，H ）所指的位址（ 105 行～107 行）。

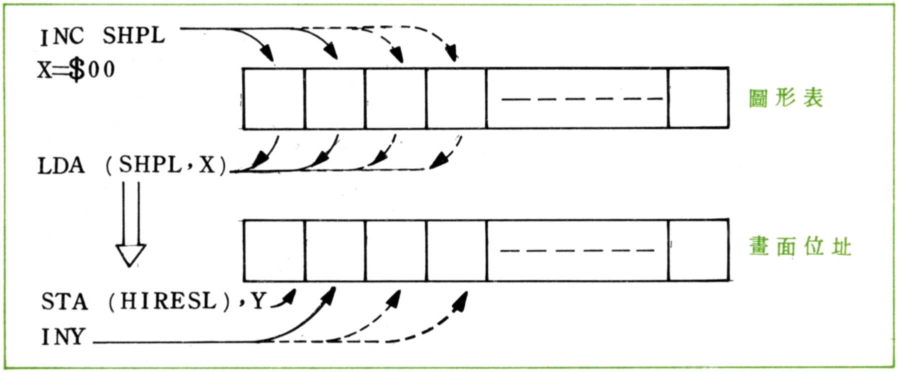


圖 4 — 2 雙指位器的資料搬動法

這種方式的好處是只要設定不同的圖形表位址 SHADR （程式 3 — 8 的53行～56行），我們仍舊可以使用 XDRAW 這段副程式來畫出不同的圖形。而其缺點是 LDA （ SHPL，X ）及 INC SHPL均是較費時的指令（共11 個工作週期）。

反過來看程式 4 — 0 ，由於67行～72行的自動位址設定方式，使得畫圖工作可以用 LDA SHTAB ，X 及INX這種快速的指令來執行（共 4 + 2 = 6 個工作週期，幾乎爲 3 — 8 的一半時間）。這是因爲 3 — 8 是以零頁位址（ SHPL，H ）做指位器，因此必須以零頁的索引間接定址法 LDA （ SHPL ，X ）來取資料，並且也因此只能用 INC SHPL而不能用 INX 來取下一個位址的資料。而 4 — 0 則直接設定SHTAB 這個絕對位址，所以用 LDA SHTAB ，X 及 INX便直接可以取得整個圖形表的資料了。譬如 Arcade Machine 或 Graphics Magician 等最新而成功的程式均採用 4

－0的方法。程式4－0將成爲本書中最重要的副程式！

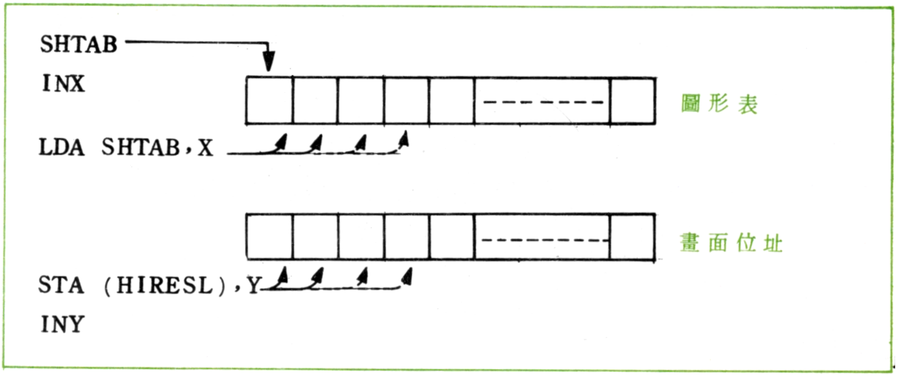


圖4－3 單指位器的圖形資料搬動法

4-3 碰撞及兩種以上圖形的處理法——走吧！大嘴巴！

程式4－0是一個相當有用的副程式，不但是效率高，速度快，並且使用起來也相當方便，只要造出預移圖序表，輸入SCX及VERT（即圖形的水平及垂直座標），程式4－0便自動可以由圖序表中找出對應的圖序來並且畫在螢幕上。所以接下來的工作重點，便是如何安排動畫情節，各個角色的造型及其動作規則了。

在4－1節當中，主程式4－1是以BASIC寫成的，這樣做有幾個缺點，首先是絕大部份的零頁位址不能任意使用，否則會破壞Applesoft Basic 的正常運作，而且程式一旦龐大，BASIC自然不能滿足速度的要求，另外是BASIC程式超過6K時，程式本身便進入到高解像畫面第一頁區的位址。所以，我們便來造一個完全由機械語言執行的動畫程式，這個程式可以處理兩個以上的不同圖形，並且能夠檢查圖形是否相互碰撞。爲了要處理多種圖形的情形，我們將程式4－0加以擴展，除了SCX及VERT之外，還加入了SHNUM（SHAPE NUMBER），如此4－0副

程式便可依照主程式的指定而畫出各個圖號（ SHNUM ）的圖形來。並且，擴展後的副程式能夠檢查畫面的底色，如果所欲畫圖的位置上已經先有圖形存在，則碰撞旗號 COLLIS≠0 ，如此使主程式得以檢查COLLIS以決定是否做碰撞處理。程式 4－2 的第 207行至 337 行便是擴展後的 DRAW 副程式，與 4－0 不同之處在於 4－2 多了一組（ SHLO，SHHI ）指位器，這是在處理多種圖形時，用以指定各個圖形的圖序表起始位址，其動作如下圖說明：

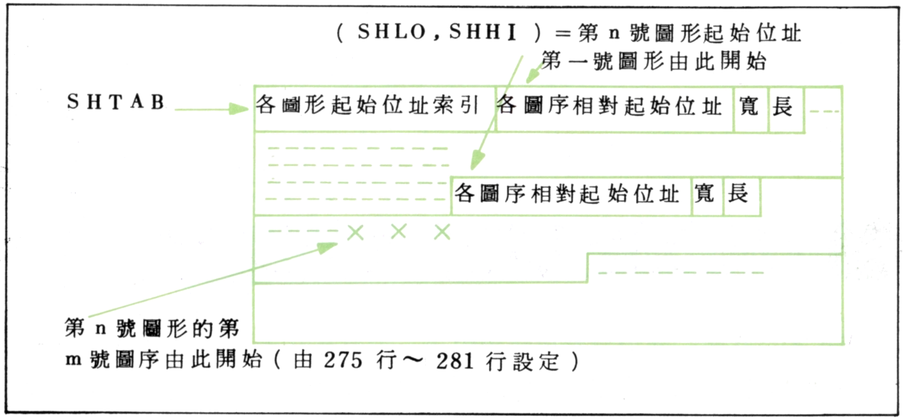


圖 4－4　多種圖形的圖序表結構

所以當主程式設定SCX、VERT 及 SHNUM 後呼叫 DRAW ，DRAW 的第 236 行至 244 行便先將SCX換算成對應的XBYTE 及 XBIT ，接著 249 行及 250 行將 COLLIS旗號置 0 ， 251 行至 258 行以 SHTAB 為起點，根據 SHNUM 找到此號圖形的起始位址，然後存入（ SHLO，SHHI ）。 266 行到 271 行取出距離（ SHLO ，SHHI ）為$0E及$0F的圖形長寬值，由 275 行至 290 行是根據 XBIT 的值找出所對應的圖序起始位址，並存入到 COLL＋1 及 COLL＋2 ，其功能與程式 4－0 的67行至72行相同，只不過前者以（ SHLO ， SHHI ）指位，而後者則是用 SHTAB 。318 行至 323 行是碰撞檢知，其中 318 行的KEY是前述的 COLL＋1 及 COLL＋2

的位置，KEY 只不過是暫時用來填充位置用的，事實上其值是由 275 ～ 290 來設定，所以任何非 0 頁的位址均可暫用於此處（因爲 0 頁位址只佔用一個位址，因此不能用）。

```
*** ANIMATION MORE !!! ***
**************************
*                        *
* Use draw routine from  *
* line 212 to line 337   *
* and shapes #0 & #1     *
*                        *
         ORG $A00
*                        *
**************************
*
* Global Variables (0-Page)
*
VERT     EPZ $02
SHNUM    EPZ $03
SCX      EPZ $04
HIRESL   EPZ $06
HIRESH   EPZ $07
COLLIS   EPZ $0B
CTLX     EPZ $F9
CTLY     EPZ $FA
SHNUM1   EPZ $FB
MAX      EPZ $FD
TEMP1    EPZ $FE
TEMP     EPZ $FF
* Non-0-page address
KEY      EQU $C000
RESETKEY EQU $C010
PREAD    EQU $FB1E
NEWX     EQU $5000
NEWY     EQU $5080
*
```

```
* CLEAR SCREEN（清除畫面）
*
START      LDA #$20
           STA HIRESH
           TAX
           LDA #$00
           STA HIRESL
           TAY
CLEAR      STA (HIRESL),Y
           INY
           BNE CLEAR
           INC HIRESH
           DEX
           BNE CLEAR
*
* ENABLE GRAPHICS
*
           STA $C050
           STA $C052
           STA $C054
           STA $C057
*
* PUT SHAPES ON SCREEN BEFORE
* START OF ERASE/REDRAW CYCLE
*（移動圖形前，先畫出圖形來）
           LDA #$00          ；圖形 0 — 5 的畫面
           STA SHNUM         ；初始設定
           LDA #$05
           STA MAX
           STA NEWX
           STA SCX
           LDA #$A0
           STA VERT
           JSR DRAW
*------SHAPE #0 IS NOW ON SCREEN
           LDX #$01
           STX SHNUM
           LDA #$20
           STA NEWX,X
           STA SCX
```

```
         JSR DRAW
*------SHAPE #1 IS NOW ON SCREEN
         LDX #$02
         STX SHNUM
         LDA #$38
         STA NEWX,X
         STA SCX
         JSR DRAW
*------SHAPE #2 IS NOW ON SCREEN
         LDX #$03
         STX SHNUM
         LDA #$50
         STA NEWX,X
         STA SCX
         JSR DRAW
*------SHAPE #3 IS NOW ON SCREEN
         LDX #$04
         STX SHNUM
         LDA #$68
         STA NEWX,X
         STA SCX
         JSR DRAW
*------SHAPE #4 IS NOW ON SCREEN
         LDX #$05
         STX SHNUM
         STX SHNUM1
         LDA #$80
         STA NEWX,X
         STA SCX
         JSR DRAW
*------SHAPE #5 IS NOW ON SCREEN
         LDX SHNUM1
         STX SHNUM
         LDA #$80
         STA CTLX
         STA SCX
         LDA #$20
         STA VERT
         STA CTLY
         JSR DRAW
```

```
*------SHAPE #0 IS NOW ON SCREEN
              LDA #$AA
              STA $2310
              STA $2710
              STA $2B10
              STA $2F10
*------A SQUARE IS NOW ON SCREEN
*
* NOW THE FOLLOWING CODE
* WILL ERASE THE SHAPE
* AT THE OLD LOCATION
* AND PUT IT AT THE NEW
* LOCATION.
* SINCE THE DRAW ROUTINE
* MAINTAINS ITS PARAMETERS
* ,NO CALCULATION IS
* NEEDED FOR DRAW/ERASE
*
XDR           LDA CTLX        ；在舊位置上消圖
              STA SCX
              LDA CTLY
              STA VERT
              JSR DRAW
              JSR GETPDLX     ；取 PDL（0）的值
              JSR GETPDLY     ；取 PDL（1）的值
DR            LDA CTLX        ；定新座標值
              STA SCX
              LDA CTLY
              STA VERT
              JSR DRAW        ；在新位置上畫圖
              LDA COLLIS      ；碰撞？
              BNE NEW         ；是，重新開始
              LDA MAX         ；廻圈設定
              STA SHNUM
              TAX
              LDA #$A0
              STA VERT
FOR           LDA NEWX,X      ；水平繞捲廻圈
              STA SCX
              JSR DRAW
```

```
                LDX  SHNUM
                INC  NEWX,X
                INC  NEWX,X
                LDA  NEWX,X
                STA  SCX
                JSR  DRAW
                DEC  SHNUM
                LDX  SHNUM
     NEXT       BPL  FOR             ；下一個
                LDA  SHNUM1
                STA  SHNUM
                LDA  #$30            ；延時
                JSR  $FCA8
     READKEY    LDA  KEY             ；讀鍵，換圖形
                CMP  #$7F
                BCC  XDR
                BIT  RESETKEY
                STA  TEMP1           ；暫存 A
                LDA  CTLX
                STA  SCX
                LDA  CTLY
                STA  VERT
                JSR  DRAW
                LDA  TEMP1           ；取回 A
                SEC
                SBC  #$B0            ；是否數字？
                CMP  MAX             ；0－MAX？
                BCS  DR
     SET        STA  SHNUM           ；是，則取該數字
                STA  SHNUM1          ；指字之圖形
                JMP  DR
     *--------------END OF SCENARIO
     NEW        JMP  START           ；從新開始
     GETPDLX    LDX  #$00            ；讀 PDL(0)
                JSR  PREAD
                CPY  #$80            ；結果存於 Y，＞$80？
                BCS  XINC2           ；是，加 2
     XDEC2      DEC  CTLX            ；否，水平減 2
                DEC  CTLX
                RTS
```

```
XINC2       INC  CTLX        ；水平加 2
            INC  CTLX
            RTS
GETPDLY     LDX  #$01        ；讀 PDL(1)
            JSR  PREAD
            CPY  #$80        ；大於 $80 ？
            BCS  YINC        ；是，垂直加 1
YDEC        DEC  CTLY        ；垂直減 1
            RTS
YINC        INC  CTLY        ；垂直加 1
            RTS
*=========END OF MAIN PROGRAM !!!
* (底下爲畫預移圖序的副程式 DRAW)
**********************************
* PROCEDURE DRAW (SCX,VERT,SHNUM)*
**********************************
*
* LOCAL VARIABLES:
*
XBYTE       EPZ  $00
XBIT        EPZ  $01
WIDTH       EPZ  $08
HEIGHT      EPZ  $09
XCOUNT      EPZ  $0A
VERT1       EPZ  $0C
SHLO        EPZ  $0D
SHHI        EPZ  $0E
*
* NON-0-PAGE ADDRESS
*
TABHI       EQU  $5100
TABLO       EQU  $51C0
********************************
* MAIN ROUTINE BEGIN HEHE !!! *
********************************
*
DRAW        LDY  #$00
            LDA  SCX
            CLC
*
```

```
* DIVIDE BY '7' ROUTINE
*
DLOOP    CMP #$07
         BCC DDONE
         SEC
         SBC #$07
         INY
         CLC
         JMP DLOOP
DDONE    STA XBIT
         STY XBYTE
*
* SET UP COLLIS FLAG
* AND SHAPE NUMBER
*
         LDA #$00
         STA COLLIS      ; 碰撞旗號歸 0
         LDA SHNUM       ; 取圖號
         ASL             ; 乘 2
         TAY
         LDA SHTAB,Y     ; 指到該圖號索
         STA SHLO        ; 引位置
         INY
         LDA SHTAB,Y
         STA SHHI
*
* (SHLO,SHHI) IS NOW POINT TO
* THE CORRESPONDING SHAPE ADDRESS
* 現在(SHLO，SHHI)指向該圖號的起始位址
*
* NOW SET UP WIDTH AND HEIGHT
*
         LDY #$0E        ; 在第 14 及 15 位
         LDA (SHLO),Y    ; 元組上取長、寬值
         STA WIDTH
         INY
         LDA (SHLO),Y
         STA HEIGHT
*
* GET SHTAB ADDRESS
```

```
*
         LDA XBIT         ; 取XBIT
         ASL              ; 乘2
         TAY
         LDA (SHLO),Y     ; 取圖形中XBIT
         TAX              ; 所對應的圖序
         INY              ; 起始位址
         LDA (SHLO),Y
*
* AUTO-MODIFY P.S.#
*
                          ; 將圖序起始
         CLC
         ADC SHLO         ; 位址存到COLL
         STA COLL+1       ; 之後的兩個位
         TXA              ; 元組上(即KEY)
         ADC SHHI
         STA COLL+2
         LDA VERT
         STA VERT1
*
* BEGIN DRAWING ONE SHAPE
*
         LDX #$00
LOOP1    LDA WIDTH
         STA XCOUNT
         LDY VERT1
         CPY #$C0
         BCC YTABLE
         LDY #$29
         BCS LOOP2
*
* LOOKING UP SCREEN ADDR TABLE
*
YTABLE   LDA TABLO,Y
         STA HIRESL
         LDA TABHI,Y
         STA HIRESH
         LDY XBYTE
*
```

```
* DRAW A LINE
*
LOOP2      CPY #$28
           BCS CONT
* ------------------SET COLLIS FLAG
COLL       LDA KEY,X
           STA TEMP
           AND (HIRESL),Y
           AND #$7F
           BEQ BLOCK1
           STA COLLIS
* ------------ STORE DATA ON SCREEN
BLOCK1     LDA TEMP
           EOR (HIRESL),Y
           STA (HIRESL),Y
CONT       INX
           INY
           DEC XCOUNT
           BNE LOOP2
* -----------------END OF DRAW LINE
           INC VERT1
           DEC HEIGHT
           BNE LOOP1
           RTS
* ----------------END OF DRAW SHAPE
SHTAB      ADR SHTAB1     ;預移圖序表位址
           ADR SHTAB1
           ADR SHTAB1
           ADR SHTAB1
           ADR SHTAB1
           ADR SHTAB0
SHTAB0     HEX 001000340058007C
           HEX 00A000C400E8030C
           HEX 0000000000007C1F
           HEX 007C1F0004100004
           HEX 1000041000041000
           HEX 0410007C1F000000
           HEX 0000000000000000
           HEX 0000783F00082000
```

```
                HEX 0820000820000820
                HEX 00783F0008200078
                HEX 3F00000000000000
                HEX 000000000000707F
                HEX 00104000707F0010
                HEX 4000104000104000
                HEX 104000707F000000
                HEX 0000000000000000
                HEX 0000607F01200001
                HEX 2000012000012000
                HEX 01200001607F0160
                HEX 7F01000000000000
                HEX 000000000000407F
                HEX 0340000240000240
                HEX 7F03400002400002
                HEX 400002407F030000
                HEX 0000000000000000
                HEX 0000007F07000104
                HEX 0001040001040001
                HEX 0400010400010400
                HEX 7F07000000000000
                HEX 000000000000007E
                HEX 0F00020800020800
                HEX 0208007E0F000208
                HEX 000208007E0F0000
                HEX 00000000
SHTAB1          HEX 00100050009000D0
                HEX 0110015001900410
                HEX 0000000000800000
                HEX A0950000A8D50000
                HEX A8C0800088008000
                HEX 8800800088008000
                HEX 8800800088008000
                HEX 88008000A8008000
                HEX A8D58000A0958000
                HEX 0000000000000000
                HEX 000000000080
                HEX 0000C0AA0000D0AA
                HEX 8100D0AA8100D0AA
                HEX 8100D0AA8100D0AA
```

```
HEX 8100D0828000D0AA
HEX 8100D0AA8100D0AA
HEX 8100D0AA8100C0AA
HEX 8000000000000000
HEX 000000000000
HEX 0080000080D58000
HEX A0D58200A0D58200
HEX A0818000A0808000
HEX A0808000A0808000
HEX A0808000A0818000
HEX A0958000A0D58200
HEX 80D5800000000000
HEX 000000000000
HEX 00000000000000AA
HEX 8100C0AA8500C0AA
HEX 8500C0AA8000C08A
HEX 8000C0828000C082
HEX 8000C08A8000C0AA
HEX 8000C0AA8100C0AA
HEX 850080AA81000000
HEX 000000000000
HEX 0000000000008000
HEX 80D4820080D58A00
HEX 80D58A0080D58000
HEX 8095800080858000
HEX 8085800080958000
HEX 80D5800080D58200
HEX 80D58A0080D48200
HEX 0000000000000000
HEX 000000000000
HEX 800000A8850000AA
HEX 950000AA9500008A
HEX 8000008280000082
HEX 8000008280000082
HEX 8000008A800000AA
HEX 810000AA950000A8
HEX 8500000000000000
HEX 000000000000
HEX 0000800000D08A00
HEX 00D4AA0000D4AA00
```

```
         HEX 00D4AA0000D4AA80
         HEX 00D4AA0000D48080
         HEX 00D4AA0000D4AA00
         HEX 00D4AA0000D4AA00
         HEX 00D08A0000000000
         HEX 00000000
         END
```

程式 4－2　多重圖形的動畫程序

```
]BR
*A00.EB0

0A00- A9 20 85 07 AA A9 00 85
0A08- 06 A8 91 06 C8 D0 FB E6
0A10- 07 CA D0 F6 8D 50 C0 8D
0A18- 52 C0 8D 54 C0 8D 57 C0
0A20- A9 00 85 03 A9 05 85 FD
0A28- 8D 00 50 85 04 A9 A0 85
0A30- 02 20 3C 0B A2 01 86 03
0A38- A9 20 9D 00 50 85 04 20
0A40- 3C 0B A2 02 86 03 A9 38
0A48- 9D 00 50 85 04 20 3C 0B
0A50- A2 03 86 03 A9 50 9D 00
0A58- 50 85 04 20 3C 0B A2 04
0A60- 86 03 A9 68 9D 00 50 85
0A68- 04 20 3C 0B A2 05 86 03
0A70- 86 FB A9 80 9D 00 50 85
0A78- 04 20 3C 0B A6 FB 86 03
0A80- A9 80 85 F9 85 04 A9 20
0A88- 85 02 85 FA 20 3C 0B A9
0A90- AA 8D 10 23 8D 10 27 8D
0A98- 10 2B 8D 10 2F A5 F9 85
0AA0- 04 A5 FA 85 02 20 3C 0B
```

```
0AA8- 20 1A 0B 20 2D 0B A5 F9
0AB0- 85 04 A5 FA 85 02 20 3C
0AB8- 0B A5 0B D0 5A A5 FD 85
0AC0- 03 AA A9 A0 85 02 BD 00
0AC8- 50 85 04 20 3C 0B A6 03
0AD0- FE 00 50 FE 00 50 BD 00
0AD8- 50 85 04 20 3C 0B C6 03
0AE0- A6 03 10 E2 A5 FB 85 03
0AE8- A9 30 20 A8 FC AD 00 C0
0AF0- C9 7F 90 A9 2C 10 C0 85
0AF8- FE A5 F9 85 04 A5 FA 85
0B00- 02 20 3C 0B A5 FE 38 E9
0B08- 81 C5 FD 30 03 4C AE 0A
0B10- 85 03 85 FB 4C AE 0A 4C
0B18- 00 0A A2 00 20 1E FB C0
0B20- 80 B0 05 C6 F9 C6 F9 60
0B28- E6 F9 E6 F9 60 A2 01 20
0B30- 1E FB C0 80 B0 03 C6 FA
0B38- 60 E6 FA 60 A0 00 A5 04
0B40- 18 C9 07 90 08 38 E9 07
0B48- C8 18 4C 41 0B 85 01 84
0B50- 00 A9 00 85 0B A5 03 0A
0B58- A8 B9 C9 0B 85 0D C8 B9
0B60- C9 0B 85 0E A0 0E B1 0D
0B68- 85 08 C8 B1 0D 85 09 A5
0B70- 01 0A A8 B1 0D AA C8 B1
0B78- 0D 18 65 0D 8D AA 0B 8A
0B80- 65 0E 8D AB 0B A5 02 85
0B88- 0C A2 00 A5 08 85 0A A4
0B90- 0C C0 C0 90 04 A0 29 B0
0B98- 0C B9 C0 51 85 06 B9 00
0BA0- 51 85 07 A4 00 C0 28 B0
0BA8- 13 BD 00 C0 85 FF 31 06
0BB0- 29 7F F0 02 85 0B A5 FF
0BB8- 51 06 91 06 E8 C8 C6 0A
0BC0- D0 E3 E6 0C C6 09 D0 C3
0BC8- 60 E1 0C E1 0C E1 0C E1
0BD0- 0C E1 0C D5 0B
```

程式 4 — 2 的機械碼（不包括圖序表）

程式 4-2由 132 行至 184 行是主程式，132行以前是初值設定工作，184 行以後是副程式，344 行以後是圖形表。而事實上一個佔用數K的 Game ，圖形表佔了¼～½，副程式佔用約½，而主程式僅是小部份而已。爲免佔篇幅，程式 4 － 2 的機械碼表只列出程式部份，而圖形表部份只要將程式 4 － 2 本身的 344 行～437 行接在程式後面鍵入便可。機械碼表的最後12個位元組 (E1，0C，E1，0C …)，其實是圖 4-4 中的圖形起始位址索引部份，如果您輸入其它的圖形（或更多的圖形），則這幾個位元組也要跟著改變。

程式執行時，畫面會出現一個小方塊，另外有一個外星人在移動，板動搖桿便可以控制外星人移動的方向，一旦大嘴巴與方塊碰撞，程式便從頭開始，畫面上看來，好像大嘴巴被彈開來一般。如果您按下 ′ 0 ′ 以外的鍵，外星人馬上換成圖 4 － 5 的大嘴巴了。這個程式雖然簡單，但却告訴我們如何處理碰撞、圖形變換及多種圖形的處理方法。而大多數的動畫結構只不過是程式 4 － 2 的推廣而已。在畫面底下繞捲的一群大嘴巴又是怎樣做出來的呢？請參考程式說明。特別要說明的是，由於使用了EOR 繪圖法（程式 4 － 2 的 326、 327 行），因此必須先畫出初始畫面，才可以進行消圖－－繪圖程序來移動圖形，因爲偶數次繪圖，圖形會消失，繪圖動作必須是奇數次才行！

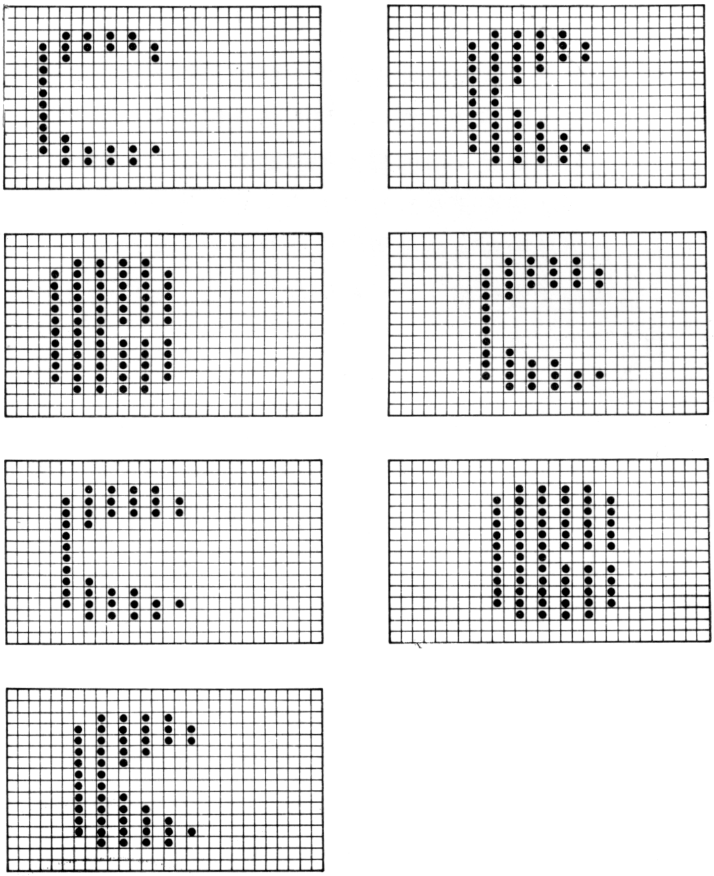


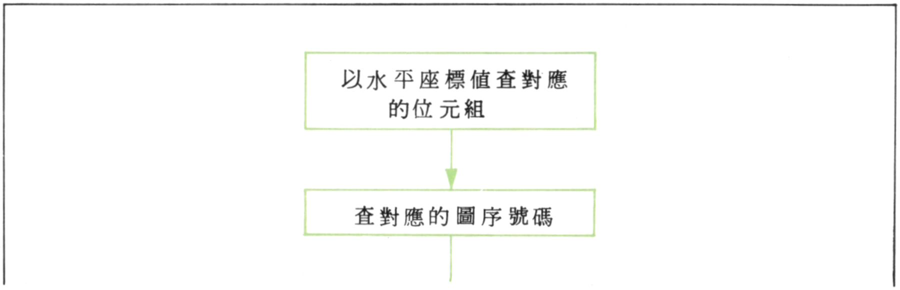
圖 4－5　大嘴巴的預移圖序

4-4 更徹底的來解決問題——完全的查表法

我們在第 0 章學到了畫面位址查表法，使得執行速度增加許多，後來在第 2 章又介紹了預移圖序法，將這兩種方法合併起來，我們便造出了如 4 － 0 這樣的快速繪圖程式，這個程式的優點是，一切的圖形資料存取均經由表格尋址，而不使用程式來計算。不過，或許您已發現到，程式 4 － 0 中，仍然有小部份使用計算的地方，那就是36至44行的 Divide by ' 7 ' 迴圈，這個程式很簡短，在一般的使用場合，並不會耗去太多時間，可說是一個合理的安排。不過動畫的複雜性及完美要求逐漸提高時，我們仍需要下工夫來將甚至是一個時脈週期的時間節省下來，所以除了垂直座標的畫面查表之外，應該也可以將水平方向的計算迴圈以表格查尋來取代吧！這樣的處理方式，在龐大的彩色圖形運動的情況下更能顯示其功效，一方面是其速度的快捷，另方面是使用表格安排的彩色圖形，似乎比複雜的計算更容易掌握而不至發生錯誤。

現在我們以彩色圖形的情況來說明水平方向位址表的使用方法：

14組圖序的水平查表法



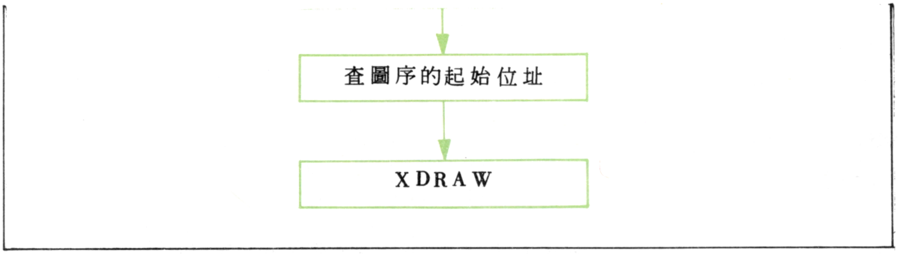


圖 4 - 6

為了要省去計算過程，因此只要定出畫圖位置（即SCX值），我們便可以經由一組表格而查出其相對應的畫面位元組及位元（亦即XBYTE 及 XBIT ）。但事實上，儘管是只在畫面上畫出一個點，其實我們也必須要將圖形資料存入到整個位元組裡，也就是說，Apple 畫面的最小運作單位是位元組而不是位元！預移圖序之產生也是基於這個原因的：由於無法在一個位元組內部做出更小的運作，因此只好以 7 個不同的圖序畫在同一個位元組上，以造出類似於細分成 7 個位元（其實是一個位元組）的運動。這便是圖 4 － 6 的流程當中沒有算出 XBIT 而直接查圖序號碼的原因了，因為即便是程式 4 － 2 的 DIVIDE′ 7 ′副程式中算出來的XBIT ，也是為了查出圖序號碼而不是用來直接畫圖的。

程式 4 － 3 便是一個例子，如果是彩色圖形的情況，則如 2 － 2 節的說明，7 個圖序位移是需要佔用兩個位元組長度的，因此程式 4 － 3 的第 34、35 行共使用了14 個圖序號碼，其中第 0 號與第 1 號，第 2 號與第 3 號，………第 12號與13號圖序是相同的，以 7 個圖序兩兩重覆來排出14個圖序號碼，而圖序製作方式則如 2 － 2 節所述。因此當由主程式控制物體沿水平座標 0 ，1 ，2 ，3 ，4 ，………運動時，圖形便以 0 ，0 ，2 ，2 ，4 ，4 ，………的位置在畫面上出現，因此彩色不受移位的影響。如果是黑白圖形的情況，則使用 7 個圖序（不重覆）便可以了。

```
**********************************
* STRATEGY-1 FOR COLOR PRE-SHIFT *
* TABLE (AN ILLUSTRATION NOT AN  *
* EXCUTEABLE PROGRAM !!!)        *
**********************************
*
SETUP     LDY XPOS              ;取畫面水平座標（ 0 — 255 ）
          LDA XBYTE,Y           ;取對應的位元組
          STA HORIZ             ;輸出到HORIZ
          LDX SHTBNUM,Y         ;取對應的圖序表號碼
          LDA SHADRL,X          ;取對應的圖序表位址
          STA SHPL              ;輸出到（ SHPL，SHPH ）
          LDA SHADRH,X
          STA SHPH
* THEN ...............
*
*
*--------------------------------*
* TABLE AREA BEGIN HERE !!!      *
*--------------------------------*
*
XBYTE     HEX 00000000000000    ;水平第 0 位元組
          HEX 01010101010101     ...............
          HEX 02020202020202     ...............
          HEX 03030303030303    ; 水平第 3 位元組
*             ..................
*             ..................
*             ................ETC.
          HEX 26262626262626
          HEX 27272727272727    ;水平第 39 位元組
*
*
SHTBNUM   HEX 00010203040506    ; 0 — 6 號圖序
          HEX 0708090A0B0C0D    ; 7 — 13 號圖序
          HEX 00010203040506    ; 0 — 6 號.........
          HEX 0708090A0B0C0D    ;...................
*             ..................
*             ..................
```

```
*              ...............ETC.
*
*
SHADRL    HEX 0024486C90B4D8    ; 0 — 6 號圖序起始位
          HEX FC2044688CB0D4    ; 7 — 13 號圖序起始位
SHADRH    HEX 00000000000000
          HEX 00010101010101
*------------END OF TABLE !
```

程式 4 - 3　畫面水平位址查表法(一)

7組圖序的水平查表法

X 座標	0 1	2 3	4 5	6 7	8 9	AB	CD	E F	10 11	12 13	14 15	16 17	…
X 位元組		0				1				2			3
X 位元	0 0	1 1	2 2	3 3	4 4	5 5	6 6	0 0					
圖形表	0	1	2	3	4	5	6	0	1	2	3	4	…

圖 4 - 7

上述的方法很直覺明瞭，但是既然第 0 號與第 1 號，第 2 號與第 3 號，………等圖序是相同的，那又何必重覆使用這麼多圖序表呢？我們可以每號圖序表只列出一個，凡是重覆使用到的，便令其找相同的圖序表便可以了，所以第 0 、 1 位置均找第 0 號圖序，第 2 、 3 位置則找第 1 號圖序，………依此類推。

不過，由圖 4 － 6 可以看出，當 X 座標為 6 的時候，是要畫第 3 號圖序的，而照我們的原則，X 座標的6 、 7 位置是要畫同樣的圖序，問題是

第 7 位置是屬於第 1 位元組的範圍而不是第 0 位元組，如果將第 3 號圖序畫到第 1 位元組內，則每 2 — 2 節圖 2 — 5 的說法(第 0 位元組的b_6是與第 1 位元組的b_0不同色)不符，因此第 7 位置仍舊是要將第 3 號圖序畫在第 0 位元組而不是第 1 位元組，這便是圖 4 — 6 及程式 4 — 4 第23、24行的由來了。

```
 2 ********************************
 3 * STRATEGY-2 FOR COLOR PRE-SHIFT *
 4 * TABLE (AN ILLUSTRATION NOT AN  *
 5 * EXCUTEABLE PROGRAM !!!)        *
 6 ********************************
 7 *
 8 SETUP      LDY XPOS                ;說明同前
 9            LDA XBYTE,Y
10            ST  HORIZ
11            LDX SHTBNUM
12            LDA SHADRL,X
13            STA SHPL
14            LDA SHADRH,X
15            STA SHPH
16 * THEN ...............
17 *
18 *
19 *-------------------------------*
20 * TABLE AREA BEGIN HERE !!!     *
21 *-------------------------------*
22 *
23 XBYTE      HEX 00000000000000      ;注意第 6 位址
24            HEX 00010101010101      ;與第 7 位址使
25            HEX 02020202020202      ;用相同圖序
26            HEX 02030303030303
27 *          ..................
28 *          ..................
29 *          ...............ETC.
30            HEX 26262626262626
31            HEX 26272727272727
```

```
*
*
SHTBNUM   HEX 00000101020203   ；每兩個位置才
          HEX 03040405050606   ；換一次圖序
          HEX 00000101020203
          HEX 03040405050606
*         ..................
*         ..................
*         ..............ETC.
*
*
SHADRL    HEX 00244860090B4D8  ；只用一半圖序
SHADRH    HEX 00000000000000
*------------END OF TABLE !
```

程式 4 - 4　畫面水平位址查表法(二)

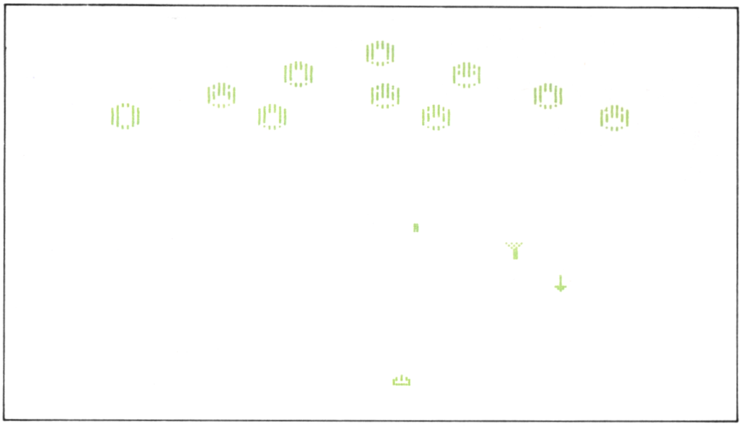
上述的說法是假設主程式每次以一個位元距離來移位的，如果是兩個位元間隔移位的情況，則參照 4 — 0 節的 1 、3 、5 、7 、2 、4 、6 方式更能節省空間。另外，再提醒您注意的是，我們在這一節當中所花的這麼多工夫，竟然只不過是為了節省Divide′ 7 ′這段廻圈的時間而已！

本章提要：

到現在，我們已將設計 Apple 動畫的各細節技巧介紹完畢，基本上我們有一個 XDRAW 副程式（見程式 4－0、4－2），因此"只要"造出預移圖序，其它的工作就很簡單了。可是，造預移圖序却是件耗時費神的工作呢！因此在本章當中：

1. 我們將介紹一個很方便的自動造表程式，使得預移圖序表對您來說再也不是件苦差事了！
2. 我們將說明 Arcade Game 的基本架構及設計方法。
3. 並且有一個〃小蜜蜂〃式的 Game 來印證我們的理論。

5 動畫設計的策略與實現(一) ARCADE GAME



如果您造過自己的預移圖序，算過格子，換算成圖序表（16進位），最後還鍵入到 Apple ………！您必然體會到這是怎樣煩複的一項工作，對於這種呆板而費時的工作，該是可以求諸電腦的幫忙吧？所以：

5-0 自動的造表程式：SHAPEDIT

這樣的程式，要怎樣著手呢？既然是造圖序表，當然是由畫圖開始，既然是畫圖，何不直接畫在 Apple 的畫面上？既然是畫在畫面上，要取得圖形資料就等於是從HIRES RAM 取值一般的容易了！所以，我們可以寫一個程式，就像一般常見的繪圖方式一樣：I 向上、J 向左、K 向右、M 向下，圖序可以一個一個分別畫（以造出本身的動作效果），也可以 7 個一次畫完（不作內部動作的圖形），然後請 Apple 將畫面的圖點以 PEEK（X）的方式取到程式內計算，最後存入到某一記憶位址區內，如此這般的一貫作業，您是不費分毫氣力的！

可是怎樣計算呢？首先一個問題是，我們是以位元的形式在畫點，但 Apple 是以位元組的形式在存取資料，所以第一步是必須將我們所畫的圖形大小換成以位元組爲單位的長寬值，對於垂直的長度是沒任何問題的，所以便來看看水平寬度是怎樣構造：

水平寬度（W）	水平位元組（BY）
1 個點	1 位元組
2 個點	2 位元組
3 個點	2 位元組
⋮	⋮
8 個點	2 位元組
9 個點	3 位元組
⋮	⋮
16 個點	4 位元組
⋮	⋮
（圖序寬度，單位：點）	（佔用的位元組）

圖 5 - 0　圖序寬度與位元組的換算

回顧我們處理預移圖序的方法，可以知道除了單一圖點寬度只用一位元組寬度外，兩個圖點寬便要佔用兩個位元組寬度（見 2-1，2 節），當圖形寬度每增加 7 圖點時（7 圖點恰佔一位元組）便要多用一個位元組，這便是圖 5－0 的由來。像圖 5－0 這種表格要如何來整理成一個公式呢？由於每

增加 7 個圖點才增加一位元組，因此其關係必然是：

BY ＝ INT((W+C)/7)…………(式 5-0)

其中ＢＹ為圖序佔用的寬度（單位為位元組），W 則為圖序實際寬度（單位為位元），C 則係一未定常數。由圖 5 — 0 可以看出：

(1) 13 ＞ C ≥ 6	；否則W＝1時	BY≒1……	(式 5-1)
(2) 18 ≥ C ≥ 12	；否則W＝2時	BY≒2……	(式 5-2)
(3) 13 ＞ C	；否則W＝8時	BY≒2……	(式 5-3)

所以唯一的解是 C ＝12，並且這個答案對於 W ＞ 8 的情況皆成立。因此：

BY ＝ INT((W+12)/7)…………(式 5-4)

知道了長寬位元組之後，圖序表的最前14個位元組（個別圖序相對於圖序表開頭的距離位元組）便可以算出來了，第 1 號圖序距離當然是14（$0E），因為是緊接在此14個距離位元組之後，而第 2 號圖序距離則是在距離第 1 號圖序BY× H ＋ 2 的地方（加 2 是因為每號圖序最開頭多加了長寬位元組的緣故），所以距離開頭共14＋ BY × H ＋ 2 位元組，H 為圖序高度其他類推：

```
0  REM ********************************
1  REM * RELATIVE ADDRESS CALCULATION *
2  REM ********************************
10 L = 21248: REM  $5300
20  FOR I = 0 TO 6: REM  7-PRESHIFT SHAPE
```

```
30 J = (BY * H + 2) * I + 14
40  POKE L, INT (J / 256)
50  POKE L + 1,J -  PEEK (L) * 256
60 L = L + 2
70  NEXT I
```

程式 5 - 0　計算相對距離的程式說明

至於眞正的圖序值，只要將畫面圖形（即HIRES RAM 值）取入到我們所要存於圖序表的位址即可，此處我們暫定圖序表存在由＄5300 開始的位址。

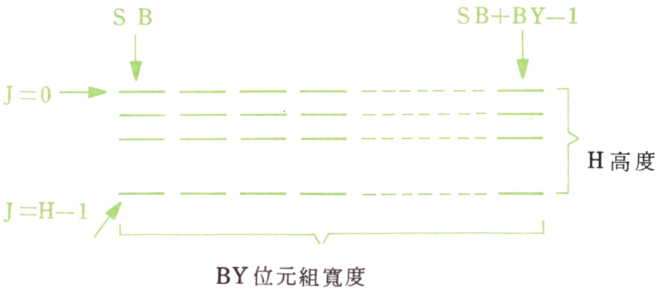


圖 5 - 1　圖形的長寬位置

第 1 位元組	第 2 位元組	位	址	內含	值
○ • • ○ ○ ○ ○	○ ○ ○ ○ ○ ○ ○	$3101	$3102	06	00
○ ○ • • ○ ○ ○	○ ○ ○ ○ ○ ○ ○	$3106	$3107	0C	00
○ ○ ○ • • ○ ○	○ ○ ○ ○ ○ ○ ○	$310B	$310C	18	00
○ ○ ○ ○ • • ○	○ ○ ○ ○ ○ ○ ○	$3110	$3111	30	00
○ ○ ○ ○ ○ • •	○ ○ ○ ○ ○ ○ ○	$3115	$3116	60	00
○ ○ ○ ○ ○ ○ •	• ○ ○ ○ ○ ○ ○	$311A	$311B	40	01
○ ○ ○ ○ ○ ○ ○	• • ○ ○ ○ ○ ○	$3120	$3121	00	03

圖 5 - 2　小球最上面一線的 7 組圖序

圖 5 - 3　小球圖序的水平座標位置（X）

由 2 - 1、2-2 節我們知道 7 個圖序的左邊起始點必須依次爲位元組的第 0 、1 、2 、3 、4 、5 、6 位元，而圖 5 － 3 的小球 X 座標值除以 7 的結果分別恰是 0 、1 、2 、3 、4 、5 、6 ，所以只要 7 個圖序同時由這些點開始畫圖的話，其相對關係正好構成一組預移圖序。程式 5 － 2 便是用來畫出在圖 5 － 3 的 X 座標上 7 個小球的簡單程式：

```
0   REM *********************
1   REM * PRESHIFT LOCATIONS *
2   REM *********************
5   REM
6   HGR
7   HCOLOR= 3
8 K = 7
10  FOR I = 0 TO 6
20  FOR H = 0 TO 3
30  FOR W = 0 TO 3
40  READ C
50  IF C THEN  HPLOT K + W,20 + H
60  NEXT W: NEXT H
65  RESTORE
66 K = K + 36: REM DISTANCE IS 35 & PRESHIFT 1
67  NEXT I
70  DATA 0,1,1,0
80  DATA 1,1,1,1
90  DATA 1,1,1,1
100  DATA 0,1,1,0
```

程式 5 - 2　畫小球預移圖序的程式

如果程式5－2可以畫出正確的圖序，則程式執行後，其畫面位址內含值必須就是我們所要的預移圖序表，請先檢示圖5－3的小球及其位置，然後將最上面一線的取出來畫出圖序及圖序表，您的結果必然如圖5－2一般。而這個結果也必須是與程式5－2的結果相同才行：

```
*3100.3127 （小球第一線）

3100-  00  06  00  00  00  00  0C  00
3108-  00  00  00  18  00  00  00  00
3110-  30  00  00  00  00  60  00  00
3118-  00  00  40  01  00  00  00  00
3120-  03  00  00  00  00  00  00  00
*3500.3527 （小球第二線）

3500-  00  0F  00  00  00  00  1E  00
3508-  00  00  00  3C  00  00  00  00
3510-  78  00  00  00  00  70  01  00
3518-  00  00  60  03  00  00  00  40
3520-  07  00  00  00  00  00  00  00
*3900.3927 （小球第三線）

3900-  00  0F  00  00  00  00  1E  00
3908-  00  00  00  3C  00  00  00  00
3910-  78  00  00  00  00  70  01  00
3918-  00  00  60  03  00  00  00  40
3920-  07  00  00  00  00  00  00  00
*3D00.3D27 （小球第四線）

3D00-  00  06  00  00  00  00  0C  00
3D08-  00  00  00  18  00  00  00  00
3D10-  30  00  00  00  00  60  00  00
3D18-  00  00  40  01  00  00  00  00
3D20-  03  00  00  00  00  00  00  00
```

* 請對照圖5－2中的位址內含值，因爲畫面位址不連續，所以本圖中的圖序值看來不連續，其實使用上，是要取畫面位址表才能使用的。

圖5－4　由程式5－2造出的預移圖序表

觀察圖5－4當中$3100至3127的位址內含值，您便可以知道，程式5－2的結果正是圖5－3的小球預移圖序表。

那麼一切便很清楚了，只要我們如圖5－3一樣的定出圖序左邊起始

位址，然後同步繪圖（同步移動圖點），當繪圖完成之後將這些畫面位址的內含值（即我們的圖序表值）PEEK 出來，再 POKE 到我們所寫程式的圖序表位址內（譬如＄ 5300），這便是自動造表程式的來龍去脈了。

5-1 造表程式及其使用法

我們設計出來的程式是這樣子的：

```
]LIST

10  REM ********************************
20  REM *    SHAPE EDITOR & COMPILER    *
30  REM *    AUTO-CREATED SHAPE TABLE   *
40  REM ********************************
45  REM
49  REM    ***------------------------***
50  REM    *** NOW BEGIN THE PROGRAM! ***
51  REM    ***------------------------***
60  HGR
99  PRINT  CHR$ (4);"BLOAD TABHI.LO"
100  PRINT  CHR$ (4);"BLOAD GAME#5.1.OBJ"
102  REM   ***------------------------***
103  REM   *** SET UP DOT SHAPE TABLE ***
104  REM   ***------------------------***
105  ROT= 0: SCALE= 1: FOR I = 768 TO 773:
      READ V: POKE I,V: NEXT : POKE 232,0:
      POKE 233,3: DATA  1,0,4,0,4,0
106  REM   ***------------------------***
107  REM   ***  SET UP LINE LOCATION  ***
108  REM   ***------------------------***
```

```
110  DIM MD%(6),LA%(55),XS%(6)
120  FOR I = 0 TO 6:MD%(I) = 1: READ XS%(I
     ): NEXT : FOR I = 0 TO 55:Y =  INT (I
      / 8):LA%(I) = 8320 + Y * 128 + (I -
     Y * 8) * 1024: NEXT
130  DATA 7,79,151,223,43,115,187
140 FR =  FRE (0): GOSUB 1000:CM = 0:SS =
     1:XC = 3:YC = 3: GOSUB 1200
145 CS = 3:HB = 0: HOME : VTAB 21: PRINT "
     I J K M Z X Q W H E O A 1-7 D": PRINT
     "HBIT OFF     ALL "
146  REM  ***-------------------------***
147  REM  *** WAITING FOR INSTRUCTION ***
148  REM  *** & FLASHING SURSOR !!!   ***
149  REM  ***-------------------------***
150  IF  PEEK ( - 16384) < 128 THEN  GOSUB
     1100: GOTO 150
160  GET A$: IF A$ = "I" AND YC > 0 THEN Y
     C = YC - 1: GOTO 280
165  IF A$ = "M" AND YC < 55 THEN YC = YC +
     1: GOTO 280
170  IF A$ = "J" AND XC > 0 THEN XC = XC -
     1: GOTO 280
175  IF A$ = "K" AND XC < 49 THEN XC = XC +
     1: GOTO 280
180  IF A$ = "Z" THEN SS = 1: GOTO 282
185  IF A$ = "X" THEN SS = 1: GOTO 284
186  IF A$ <  > "H" THEN 200
190 HB = 4 - HB: VTAB 22: HTAB 1: IF HB =
     0 THEN  PRINT "HBIT OFF": GOTO 150
195  PRINT "HBIT ON ": GOTO 150
200  IF A$ > "0" AND A$ < "8" THEN MD%( VAL
     (A$) - 1) = 1 - MD%( VAL (A$) - 1): GOSUB
     1000: GOTO 150
```

```
205  IF A$ = "Q" THEN SS = 2: GOTO 282
210  IF A$ = "W" THEN SS = 3: GOTO 284
215  IF A$ = "A" THEN CS = 3: VTAB 22: HTAB
     13: PRINT "ALL ": GOTO 150
220  IF A$ = "E" THEN CS = 2: VTAB 22: HTAB
     13: PRINT "EVEN": GOTO 150
225  IF A$ = "O" THEN CS = 1: VTAB 22: HTAB
     13: PRINT "ODD ": GOTO 150
230  IF A$ <  > "D" THEN 150
232  REM  ***-------------------------***
233  REM  ***          DONE !!!        ***
234  REM  ***-------------------------***
235  VTAB 23: HTAB 1: PRINT "E C A S L N
      :";: GET A$: PRINT A$: IF A$ = "E" THEN
      GOSUB 1300: GOTO 145
240  IF A$ = "C" THEN  GOSUB 2000:CM = 1: GOTO
     235
242  REM  ***-------------------------***
243  REM  ***     SAVE SHAPE TABLE     ***
244  REM  ***-------------------------***
245  IF A$ <  > "S" THEN 250
246  IF  NOT CM THEN 235
247  GOSUB 1300: INPUT "NAME:";A$: ONERR  GOTO
     249
248  PRINT  CHR$ (4);"BSAVE";A$;".SSH,A$53
     00,L";SL: POKE 216,0: GOTO 235
249  GOSUB 1300: PRINT "DISK ERROR.": GET
     A$: POKE 216,0: GOTO 235
250  IF A$ <  > "L" THEN 260
251  REM  ***-------------------------***
252  REM  ***     LOAD SHAPE TABLE     ***
253  REM  ***-------------------------***
254  GOSUB 1300: INPUT "NAME:";A$: ONERR  GOTO
     249
```

```
255  PRINT  CHR$ (4);"BLOAD";A$;".SSH,A$53
     00,": POKE 3,0: POKE 4,0: POKE 216,0:
      HGR : GOSUB 1000:CM = 0: POKE 2,8: FOR
     I = 0 TO 3: GOSUB 258: NEXT : POKE 2,
     72: FOR I = 4 TO 6: GOSUB 258: NEXT :
      GOTO 235
258  POKE 4,XS%(I): CALL 7943: RETURN
259  REM  ***-----------------------***
          ***  ANIMATION  ROUTINE!  ***
          ***    USING PADDLE TO    ***
          ***     CONTROL SPEED     ***
          ***-----------------------***
260  IF A$ <  > "A" THEN 270
261  POKE  - 16302,0: POKE 2,160: POKE 4,1
     0: CALL 7943
262  CALL 7936: FOR I = 1 TO 510 -  PDL (0
     ) * 2: NEXT : IF  PEEK ( - 16384) < 1
     28 THEN 262
264  CALL 7943: POKE  - 16301,0: GOTO 235
265  REM  ***-------------------------***
266  REM  ***   RE-START THE SYSTEM   ***
267  REM  ***-------------------------***
270  IF A$ = "N" THEN  GOSUB 350: IF I THEN
      PRINT "RUNSHAPEDIT,"
276  GOTO 235
277  REM  ***-------------------------***
278  REM  ***  'MOVE' ROUTINE HANDLER ***
279  REM  ***-------------------------***
280  GOSUB 1200: ON SS GOTO 150,282,284
282  HCOLOR= HB + CS: GOSUB 1150: GOTO 150

284  HCOLOR= HB: GOSUB 1150: GOTO 150
350  PRINT "LOSE CURRENT SHAPE?": GET A$: PRINT
     : HOME :I = (A$ = "Y"): RETURN
```

```
996   REM  ***--------------------------***
997   REM  *** FLAG  SHAPES  IN  EDIT  ***
998   REM  *** MODE  &  DRAW  7  LINES ***
999   REM  ***--------------------------***
1000  FOR I = 0 TO 6:Y = (I > 3) * 64 + 3:
      HCOLOR= 4 + MD%(I): HPLOT XS%(I),Y TO
     XS%(I) + 49,Y
1001  NEXT : RETURN
1097  REM ***--------------------------***
1098  REM ***     FLSAH    CURSORS     ***
1099  REM ***--------------------------***
1100  FOR J = 1 TO 2: FOR I = 0 TO 6: IF M
     D%(I) THEN X = XS%(I) + XC:Y = (I > 3
     ) * 64 + 8 + YC: XDRAW 1 AT X,Y
1101  NEXT : NEXT : RETURN
1147  REM ***--------------------------***
1148  REM ***  PLOT  A  DOT  (Z OR X) ***
1149  REM ***--------------------------***
1150  FOR I = 0 TO 6: IF MD%(I) THEN X = X
     S%(I) + XC:Y = (I > 3) * 64 + 8 + YC:
      HPLOT X,Y
1151  NEXT : RETURN
1199  REM  PRINT LOC.
1200  VTAB 22: HTAB 20: PRINT "X:";XC + 1;
     " Y:";YC + 1;"   ": RETURN
1297  REM ***--------------------------***
1298  REM ***  CLEAR  BELOW  CURSORS   ***
1299  REM ***--------------------------***
1300  VTAB 23: HTAB 1: CALL  - 958: VTAB 2
     3: HTAB 1: RETURN
1997  REM ***--------------------------***
1998  REM ***  COMPILER ROUTINE !!!    ***
[illegible]  REM ***--------------------------***
```

```
2000  GOSUB 1300: INPUT "WIDTH:";W: IF W <
      1 OR W > 50 THEN 2000
2002  GOSUB 1300: INPUT "HEIGHT:";H: IF H <
      1 OR H > 55 THEN 2002
2004  GOSUB 1300:BY =  INT ((W + 12) / 7):
      LN = BY * H + 2: PRINT "WIDTH:";BY;"
      BYTES, HEIGHT:";H: PRINT "SHAPE LENGT
      H:";LN - 2: IF LN > 256 THEN  PRINT "
      TOO LONG (254 MAX).": GET A$: GOSUB 1
      300: RETURN
2006  FLASH : PRINT "COMPILING": NORMAL
2010 BY =  INT ((W + 12) / 7):LN = BY * H +
      2:L = 21248: FOR I = 0 TO 6:OF = 14 +
      I * LN: POKE L, INT (OF / 256): POKE
      L + 1,OF -  PEEK (L) * 256:L = L + 2:
       NEXT
2020  FOR I = 0 TO 6: POKE L,BY: POKE L +
      1,H:L = L + 2
2025  FOR J = 0 TO H - 1:Y = LA%(J) + (I -
       INT (I / 2) * 2) * 40
2030 SB = Y + I * 5 + 1: FOR K = SB TO SB +
      BY - 1: POKE L, PEEK (K):L = L + 1: NEXT
      : NEXT : NEXT :SL = LN * 7 + 14: GOSUB
      1300: RETURN
2998  REM ***------------------------***
2999  REM ***  END  OF  PROGRAM !!!   ***
3000  REM ***------------------------***
```

程式 5-3 自動造表程式

```
**************************
* DRAWING PRE-SHIFT GRAPH *
* USING  LOOK-UP TABLE !! *
**************************
*
         ORG $1F00
*
* O-PAGE ADDRSEE
*
XBYTE    EPZ $00
XBIT     EPZ $01
VERT     EPZ $02
XCOUNT   EPZ $03
SCX      EPZ $04
HIRESL   EPZ $06
HIRESH   EPZ $07
WIDTH    EPZ $08
HEIGHT   EPZ $09
VERT1    EPZ $0C
NEWX     EPZ $FD
*
* NON-O-PAGE ADDRESS
*
TABHI    EQU $5100
TABLO    EQU $51C0
SHTAB    EQU $5300
*
*******************************
* MAIN PROGRAM BEGIN HERE !!! *
*******************************
*
DX       JSR DRAW
         LDA NEWX
         STA SCX
DR       JSR DRAW
         INC NEWX
         INC NEWX
         RTS
*******************************
* MAIN ROUTINE BEGIN HEHE !!! *
*******************************
*
DRAW     LDY #$00
         LDA SCX
         CLC
```

```
DLOOP     CMP  #$07
          BCC  DDONE
          SEC
          SBC  #$07
          INY
          CLC
          JMP  DLOOP
DDONE     STA  XBIT
          STY  XBYTE
*
* BEGIN SET-UP
*
          LDA  VERT
          STA  VERT1
*
* GET WIDTH & HEIGHT VALUE
*
          LDY  #$0E
          LDA  SHTAB,Y
          STA  WIDTH
          INY
          LDA  SHTAB,Y
          STA  HEIGHT
*
* AUTO MODIFY P.S.#
*
          LDA  XBIT
          ASL
          TAY
          LDA  SHTAB,Y
          TAX
          INY
          LDA  SHTAB,Y
          CLC
          ADC  #SHTAB
          STA  BLOCK1+1
          TXA
          ADC  /SHTAB
          STA  BLOCK1+2
* -----------END OF SET-UP
          LDX  #$02
LOOP1     LDA  WIDTH
          STA  XCOUNT
          LDY  VERT1
          CPY  #$C0
```

```
                BCC  YTABLE
                LDY  #$29
                BCS  LOOP2
YTABLE          LDA  TABLO,Y
                STA  HIRESL
                LDA  TABHI,Y
                STA  HIRESH
                LDY  XBYTE
LOOP2           CPY  #$28
                BCS  CONT
BLOCK1          LDA  SHTAB,X
                STA  (HIRESL),Y
CONT            INX
                INY
                DEC  XCOUNT
                BNE  LOOP2
                INC  VERT1
                DEC  HEIGHT
                BNE  LOOP1
                RTS
                END
```

程式 5 - 4　GAME# 5.1. ASM

```
*1F00.1F7E

1F00- 20 0F 1F A5 FD 85 04 20
1F08- 0F 1F E6 FD E6 FD 60 A0
1F10- 00 A5 04 18 C9 07 90 08
1F18- 38 E9 07 C8 18 4C 14 1F
1F20- 85 01 84 00 A5 02 85 0C
1F28- A0 0E B9 00 53 85 08 C8
1F30- B9 00 53 85 09 A5 01 0A
1F38- A8 B9 00 53 AA C8 B9 00
1F40- 53 18 69 00 8D 6E 1F 8A
1F48- 69 53 8D 6F 1F A2 02 A5
1F50- 08 85 03 A4 0C C0 C0 90
1F58- 04 A0 29 B0 0C B9 C0 51
1F60- 85 06 B9 00 51 85 07 A4
1F68- 00 C0 28 B0 05 BD 00 53
1F70- 91 06 E8 C8 C6 03 D0 F1
1F78- E6 0C C6 09 D0 D1 60
```

*請以GAME # 5.1. OBJ 存入磁片

程式 5 - 4 的機械碼

程式由第99行開始執行，99 行及100 行是將畫面位址表 TABHI.LO（由程式 0 － 2 造出）及一稱爲GAME #5.1.OBJ（即程式 5 － 4 ）的機械語言副程式由磁片載入到 APPLE 來，對於沒有磁碟機的讀者，您可以直接鍵入程式 5 － 4 的機械碼表，並且將程式 0 － 2 附加到本程式的前頭即可（99行及 100 行去掉）。這個程式可以令我們用按鍵直接在 Apple 的高解像畫面上直接造出圖形的 7 個預移圖序來，並當圖序造好後可以令 Apple 自動替我們將 7 個圖序圖換算成圖序表，然後存入磁片上。所以您可以直接在畫面上將圖形畫好，然後不用一點一點的換算，更不用將圖序表一一鍵入，一切均由 Apple 自動爲我們做好！等到將圖形編譯（Compile ）成圖序表之後，您還可以按下′A′鍵（ Animation ），這時您所造好的圖形便會活生生的在畫面上移動，如果圖形移動的效果（譬如：人物的手、脚、眼睛的動作）不符您的要求，按下任何鍵回到 Command 狀態，再按下′E′鍵便又進入圖形編輯狀態而可以修改 7 個圖序的動作了。在 Command 狀態時您還可以鍵入′L′，將以前存入到磁片上的圖形的 7 個圖序取回到畫面上來加以修改。尤其有用的是，你可以選定 1 ～ 7 個圖序中的幾個來畫圖，譬如在程式執行後，按下 1 、 3 、 5 、 7 ，則第 1 、 3 、 5 、 7 號圖序便停止作用，所以您的按鍵（ I 、 J 、K 、M ）結果便只出現在 2 、 4 、 6 號圖序上，假設這時您連續用 K 、M 鍵造出“＼”圖形，接著按下 1 、 3 、 5 、 7 ，則第 1 、 3 、 5 、 7 號圖序又開始可以畫圖了，但是第 2 、 4 、 6 號圖序已經畫完成，所以您要按 2 、 4 、 6 等字鍵使其圖序凍結。如果這時您以 J 、M 鍵造出“／”圖形，那麼可以按下 D（ Done ）鍵，告訴 Apple 繪圖完畢。此時螢幕底下的字幕會出現“E C A S L N”等字幕，你必須選擇 C 來將圖序換算（ Compiling ）成圖序表，當按下 C 之後，Apple 會問您圖序的寬度（ WIDTH ）及高度（ HEIGHT ），您可以在繪圖工作的最後將游標移到圖形的最右下方，這時螢幕底下所顯示的 X 及 Y 值便是寬度及高度值了。您只要將這兩個值回答給 APPLE ，圖序表換算工作便自動進行了。當 Compiling 完畢後，請按″A″鍵，結果如何呢？改變搖桿位置

又如何呢？

那麼換算結果的圖序表在那裡呢？圖序表的起點是設定成由＄5300開始的，您可以將其列出並根據其長寬位元組來找出其終點，計算的公式是：

總長度＝14＋（長×寬＋2）×7 ………… （式5－5）
（10進位）

您當然可以將這個公式也附加到程式上，所以，一切工作便簡化許多了。

另外對於彩色的圖形，本程式也提供相當方便的用法，在繪圖的狀態下（E　MODE　），若在HBIT OFF時按下〞H〞，則HBIT變爲ON，反之則變爲OFF。當按下〞E〞則在偶數（EVEN）座標位置上畫點，按下〞O〞則在奇數（ODD）座標位置上畫點。再配合上第一章的彩色原則，你便可以爲您的圖形配上各樣的彩色了。

自動造表程式使用簡表

程式一執行便進入繪圖狀態，此時：

I、J、K、M鍵：令游標往上、左、右、下移動。

Q鍵：按下Q鍵之後，游標移過的位置便留下圖形，除非按下W、Z或X鍵，否則Q狀態一直保持。

W鍵：按下W鍵之後，游標移過的位置不留下任何圖形，除非按下Q、Z或X鍵，否則W狀態一直保持。

Z鍵：按下Z鍵，便在目前游標所在處畫出一點來。

X鍵：按下X鍵，則目前游標所在處便不畫點。

A鍵：令所有（ALL）座標位置的點皆可畫點（視Q、W、Z、X而定）。

E鍵：令所有偶數（EVEN）水平座標位置可以畫點（當然要視Q、W、Z、X而定），奇數位置不能畫點。

O鍵：令所有奇數（ODD）水平座標位置可以畫點（視Q、W、Z、X而定），偶數位置不能畫點。

H鍵：反轉畫面位元組最高位元的狀態（B_7），因此可以改變顏色。

1-7鍵：反轉圖序設定狀態。程式執行時，7個圖序同時繪圖，按下1～7中任意一個（或多個）號碼，該圖序便凍結繪圖，再次按下該號碼便可恢復該圖序的繪圖工作了。

D鍵：結束繪圖狀態，進入命令狀態（Command Mode）。

圖5-5 (a) 繪圖指令表

在繪圖狀態時，按下'D'鍵便進入了命令狀態：

C鍵：將所繪製完畢的圖序換算成圖形表，並存入到由$5300 開始的位址內。按下C之後，您必須回答圖形的長寬值。您可以在不影響圖形的情況下（E MODE），將游標移動到圖形的右下角，此時X、Y值便是長寬值了。

S鍵：將圖序表值存入磁片當中，您必須給予檔案名稱，程式會自動在你的名稱之後接上'.SSH'，所以眞正在磁片中的檔案名稱是'XX.SSH'的樣子（SSH指的是預移圖序）。

L鍵：將磁片中的圖序取回並顯示於畫面上，以便修改或觀察。

N鍵：清除目前圖形，重新（NEW）繪圖。

E鍵：回到繪圖狀態。

A鍵：當目前所繪圖形，經由換算（Compile，即按C鍵）後，可以按下A鍵（Animation）使圖形在畫面上移動，以觀察其動態效果。改變搖桿位置可以控制移動的速度。

圖5-5 (b) Command 指令表

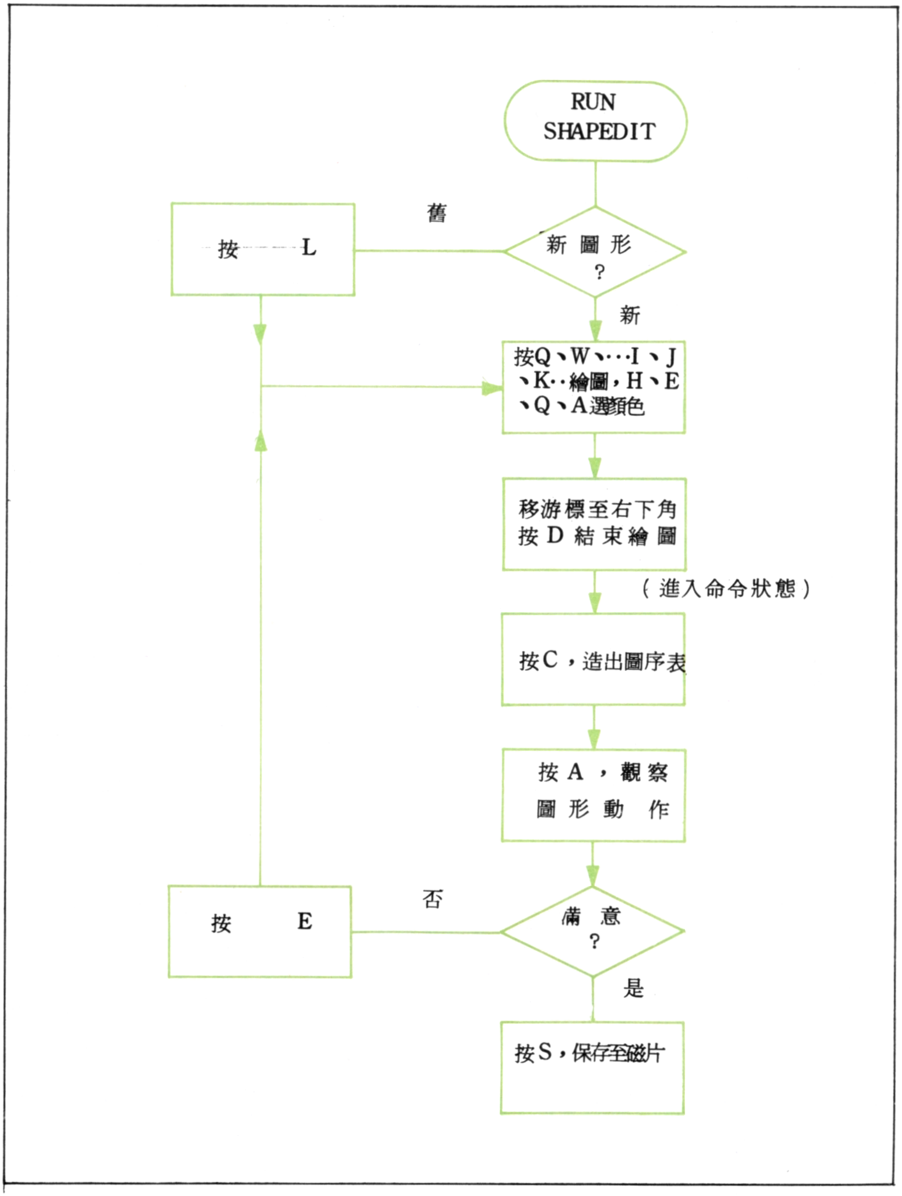


圖 5 - 6 造表程式使用流程

5-2 動畫程式設計的策略

在瞭解了繪圖動作的基本結構後，要設計一個完整的程式已經不再如想像般的困難了。多半令我們產生困擾的程式，往往在於程式整體結構的問題，而不是技巧上的細節，而這樣的程式若沒有適當的策劃，即便是用 BASIC 也一樣難以解決的。所以，我們來看看一個動畫程式應該是怎樣的規劃才較為適當。

程式是一個大迴路

除了固定背景、標題及一些裝飾之外，一個動畫程式可以說是一個無休止的迴路，只要程式一執行，迴路便一再的重覆。當然迴路並不是死板板的把每一件事在畫面上一幕一幕的演出，這其中自然是有選擇性的變化，譬如外星人會被火箭擊落而從此消失，防衛者的太空船被擊中也是粉碎不能復原，但是無論如何，在故事結束之前，整個程式是繞著一個大迴圈在跑的。所以，要設計一個動畫程式，便等於是如何將故事裡的每一角色、事件，組合成一個有意義的迴圈。底下是一些有益的意見：

1.雖然看起來，畫面上的每一個人物都在忙碌的做各種動作，其實 Apple 每一瞬間只能做一件事而已，因此往往一個程式好壞的先天因素便決定於程式本身的執行速率（效率）。一個快速有效的程式，我們可以隨時加入更多的情節變化，而不會影響故事的進行，反之，一個速率太慢的程式，加入任何額外的動作便增加了其負擔，而使畫面看起來不自然，甚至缺陷百出。

2.通常在設計的第一步是寫下故事的劇本，有那些事件、人物、在何

時、何種情況會發生，然後將劇本劃成流程圖，再一步一步來將每一流程單元實現。許多程式設計者喜歡以讀取遊戲控制器做爲程式的開頭，似乎這樣較易對程式的全盤的控制。

3.常出現的動作不要寫成副程式，這句話乍聽之下有時會覺得不合常理，其實我們在 0 － 2 節就已提到，這經常是影響效率的因素。

4.寫程式沒有一成不變的規則，許多著名的Game 彼此之間的結構差異何其之大，不過只要您詳細的研究，便會發現到一些共同的特性：效率（時間）、結構清楚、以單純的基本單元來組合出複雜而完整的程式。所以，將每個流程單元先行寫好、除錯，每個單元最好只有一個輸入點、一個輸出點，要做分支的部門特別自成一單元，如此當可組合成簡捷有效率的程式，切記莫打自己都迷糊的仗。

最佳的學習便是動手去做

所以，開始來嘗試設計動畫程式吧！以 BASIC 設計程式的人常會奇怪，爲何Apple Game的畫面可以同時處理那麼多事物的動作呢？其實這只是幻覺而已，一切歸功於機械語言的迅速有效！一大串的程式廻路可以在轉瞬之間執行完畢，所以一切就如電影般的，只是一種幻覺而已。而我們最主要的工作便是如何設計一個能夠造出這種幻覺的程式。

問題是，要處理許許多多的事物，我們怎麼樣來知道那一個角色跑到何處而又發生了何事呢？解決之道是將每一狀況、角色的資料有秩序的排列起來而存到某一位址區。譬如，我們的程式需要下列的資料：X 座標、Y 座標、下一位置的X 座標、下一位置的Y 座標、X 方向的速度、Y 方向的速度、圖形號碼，假設每一角色均有 7 個參數要依次地排列存入記憶位址裡頭，我們可以把它們以一連串的程式來完成：

```
*****************************
* EXAMPLE OF VARIABLE FILE *
* BY INDEX ADDRESSING       *
*****************************
*
OLDX        EQU $5000
OLDY        EQU $5001
NEWX        EQU $5002
NEWY        EQU $5003
VX          EQU $5004
VY          EQU $5005
FLAG        EQU $5006
*..................
*..................
SHNUM       EPZ $F0
TEMP1       EPZ $F1
*..................
*..................
* INDEX ADDRESSING ROUTINE *
* THIS IS NOT A WORKING     *
* PROGRAM, ONLY AN EXAMPLE *
*
            LDA SHNUM
            STA TEMP1
LOOP        LDA TEMP
            ASL
            ASL
            ASL
            TAX
*
* CALCULATE NEW LOCATION *
*
            LDA OLDX,X
            CLC
            ADC VX,X
            STA NEWX,X
            LDA OLDY,X
            CLC
            ADC VY,X
            STA NEWY,X
*
**************************
* XDRAW (OLDX,OLDY)      *
* XDRAW (NEWX,NEWY)      *
```

```
* AND ANY OTHER ACTIONS   *
* ALL ITEMS IN THIS LIST  *
* SHOULD BE HANDLED       *
* WITHIN THIS LOOP        *
***************************
*
         DEC TEMP
         BPL LOOP
*
***************************
* NOTE: IF OBJECTS HAVE A *
* STARTING VALUE GREATER  *
* THAN $80(B7=1) THEN     *
* BPL CANNOT BE USED      *
***************************
```

程式 5 - 5　主程式的結構說明

程式 5 — 5 以 X 做爲進入列表存取資料的索引暫存器，通常列表存放的記憶位址是以 2^n 來劃分區段，現在我們共有 7 個數據，因此只要 $2^3=8$ 個位址成一區段便可以了。譬如第一個圖形的資料由＄5000 開始存放，則只塡到＄5006 便已存完資料了，這時＄5007 不要放資料，接著由＄5008存放第二個圖形的資料，依此類推，如果數據超過 8 個，便要用 $2^4=16$個位址劃分成一區段了。這樣存放資料的好處是我們只要取得所要資料在列表中的次序，乘以 2^n 之後便可以以第一組數據位址做索引取址而取得資料。譬如程式 5 — 5 當中我們要第 3 組資料，只要將 3 存入 A 暫存器乘以 8（ ASL 3 次）然後存入 X，以 LDA OLDX，X 便可以取到在＄5000 ＋$18＝＄5018 的第 3 組資料的OLDX值，同樣的 LDA NEWY，X 便可以取到＄5003 （ NEWY 定爲＄5003 ）＋$18＝＄501B 的第 3 組資料的 NEWY 值。當然，您也可以依 0-3 節的說法將每一變數組分別存放，如此可以省下ASL等動作時間。

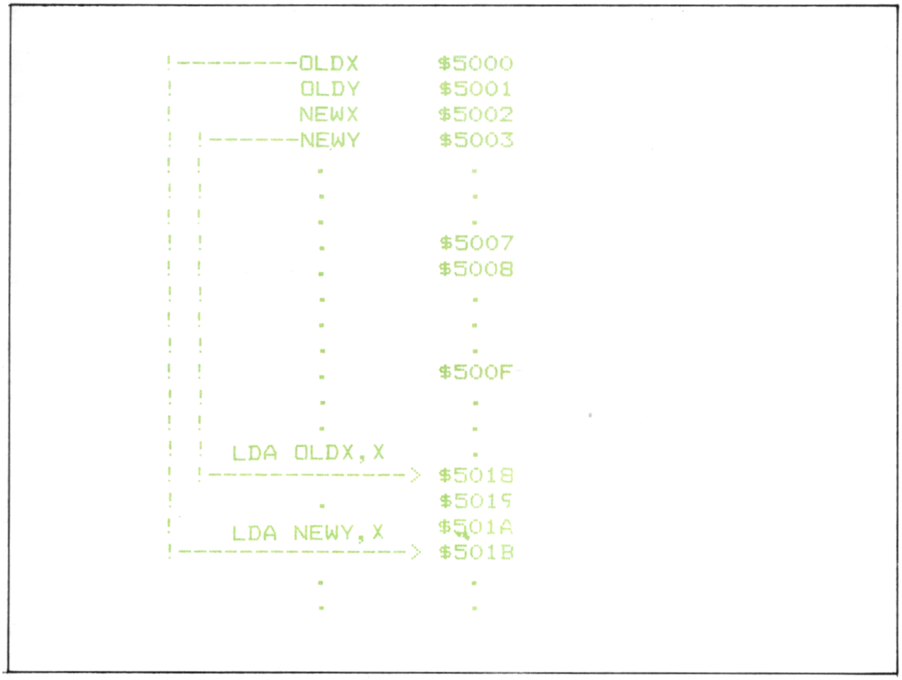


圖 5 - 7 (A) 變數排列情形

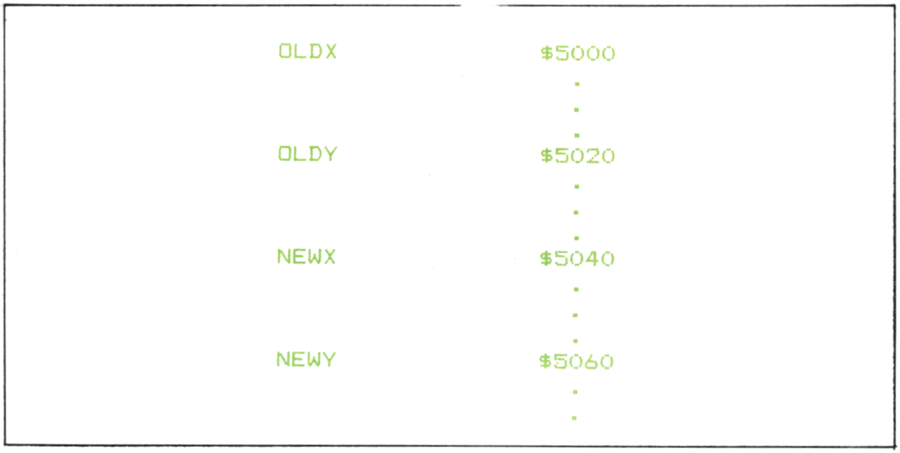


圖 5 - 7 (B) 變數組分別存放的情形

程式 5 — 5 是一個常見的動畫主程式結構，第22行至28行執行上述的取址動作，32行至39行更新這些數據的值（即位置、速度等），接著便開始作一整個遊戲廻路的動作，然後再回到前頭重覆取址、更新、………等動作，這便是畫面上各個角色運動的原理，當然在一整個廻路當中，一切碰撞檢知、按鍵或搖桿、按鈕等動作都一一檢知了，這些資料也用來做爲上述數據（ NEWX，VY ， SHAPE# ，……等）更新的依據，若有外星人被擊中，某一 SHAPE 便消除，大嘴巴移動到某處（搖桿移動）小精靈便向此方向集中，大致上便是如此。

至於圖序表、畫面位址表等一些常數資料，最好以固定位址（譬如：$ 5300 ～或$ 5100 ～等）來存放，其優點是可以直接由造表程式（ 0 — 2 或 5 — 3 ）造出之後，直接搬動到這些位址內，而不必再一一鍵入到組合語言程式內，譬如本書的例子在測試時，均是將畫面位址表造出，當成 2 進位檔案 BSAVE 到磁片內，當主程式要執行時再 BLOAD 到RAM位址裡。如果是多個圖序表的場合，那麼就要仔細計算彼此的起始及終止位址，以免相重疊了，但是對於較簡短的程式及圖序表，則以HEX資料形態鍵入到組合語言主程式內，使用上較爲方便，其缺點是佔用每次編譯（ASM）的時間。

5 3 電玩的程式設計

有關電玩（ Video Arcade ）遊戲的設計，可說是Apple 最吸引人也最令人興奮的一項挑戰，但是面對著數百數千行的組合語言程式，卻總是令人望之卻步而無從著手。可是，事實上又如何呢？

其實對我們說來，用組合語言來寫電玩程式已不是什麼困難的事情了，甚至是件比用 Applesoft 更容易的事情呢！因爲對一般人而言，BASIC的好處是其便利使用的指令，在遊戲繪圖方面這些指令就是PLOT 、 HPLOT 、 COLOR 、 HCOLOR 、XDRAW 等等。可是對於已

閱讀過前面 5 章的我們來說，BASIC是否眞的那樣方便呢？用 PLOT 、 HPLOT 來畫圖是很麻煩的事情，而 XDRAW 却只能畫單一顏色的圖形（紅眼睛、白翅膀的外星人怎麼辦？），而且畫實心的物體也是 XDRAW 的一大難題，建立 Applesoft 的圖形表可更是麻複的工作！

所以請看看我們的造形程式（5－3）吧！我們不但可以造出更快速、多重彩色、多重動作的圖形，並且不必任何計算、不必動手輸入圖形資料，便可以造出整組的預移圖序表，而只要使用如程式 4－0、4－2、等當中的 DRAW（其實是 XDRAW ）副程式，便可以快速的繪出圖形來，一切繪圖（消圖）動作只不過是：

```
LDA   #$NN
STA   SHNUM
LDA   #$XX
STA   SCX
LDA   #$YY
STA   VERT
JSR   DRAW
```

便可以在畫面座標（XX，YY）上畫出第NN號的圖形來了。對於像大嘴巴那樣的動態圖形，我們的方法是遠比BASIC 方便也有效多了。因此，一切便如 5－2 節所說的一般，當然還有您的幻想力了。

典型的電玩程式可以小蜜蜂及小精靈爲代表，而無論是對初學者或專業的程式設計者來說，前者是比後者更具學習意義及挑戰性的（後述），但是後者却因其故事情節及畫面安排的創作力而獲得相當大的成功，可見一個程式的優劣，除却程式技巧之外，更重要的是其本身的題材，也就是作者的想像力吧！所以我們便以小蜜蜂這類程式爲例來做說明吧！

動作的順序

一個動畫程式的成敗，在於其主程式的安排，而各種安排則以動作順序為優先考慮，對於小蜜蜂這種程式，我們的基本構想是：

1. 畫面上方有一群外星人，這些外星人可以在畫面上移動（若不移動則等於打磚塊之類的程式了），除了移動之外，我們還要求外星人本身也做動作，所以預移圖序的功能便得以發揮了。
2. 畫面底線有一艘可以由遊戲者控制水平移動的太空船。
3. 遊戲者按下按鈕，太空船便對外星人發射子彈。
4. 子彈打中外星人，便產生爆炸的火花，喇叭發出聲音，然後子彈及外星人皆由畫面消失。
5. 外星人會瞄準太空船的位置發射子彈，太空船若被外星人子彈打中，遊戲便結束或重新開始。

以上便是簡化了小蜜蜂程式的劇情，這裡頭的幾個要點是：

1. 如何檢知子彈擊中目標物？
2. 擊中的是哪一號目標物？
3. 如何處理（繪圖、消圖）數量龐大的外星人，而又不影響畫面速度。

某些設計者的說法是：

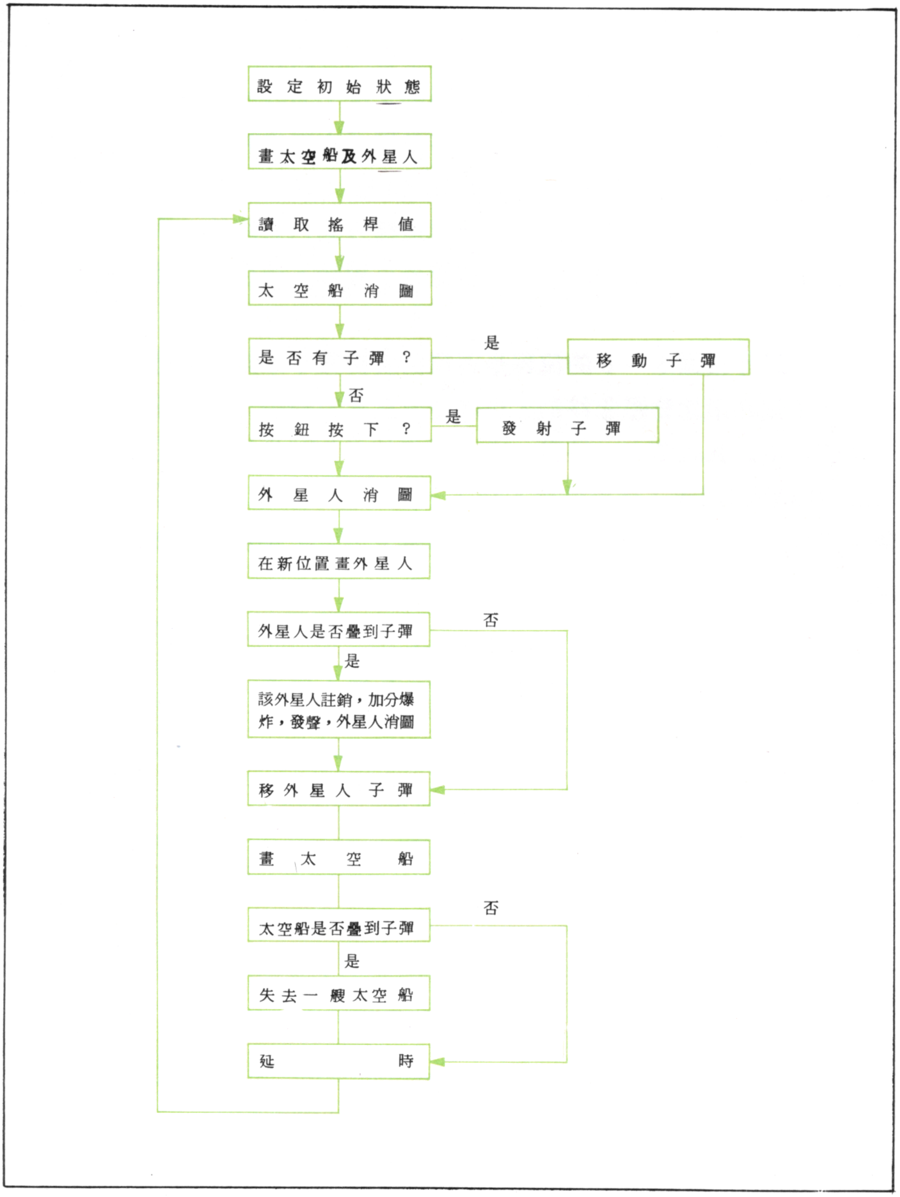
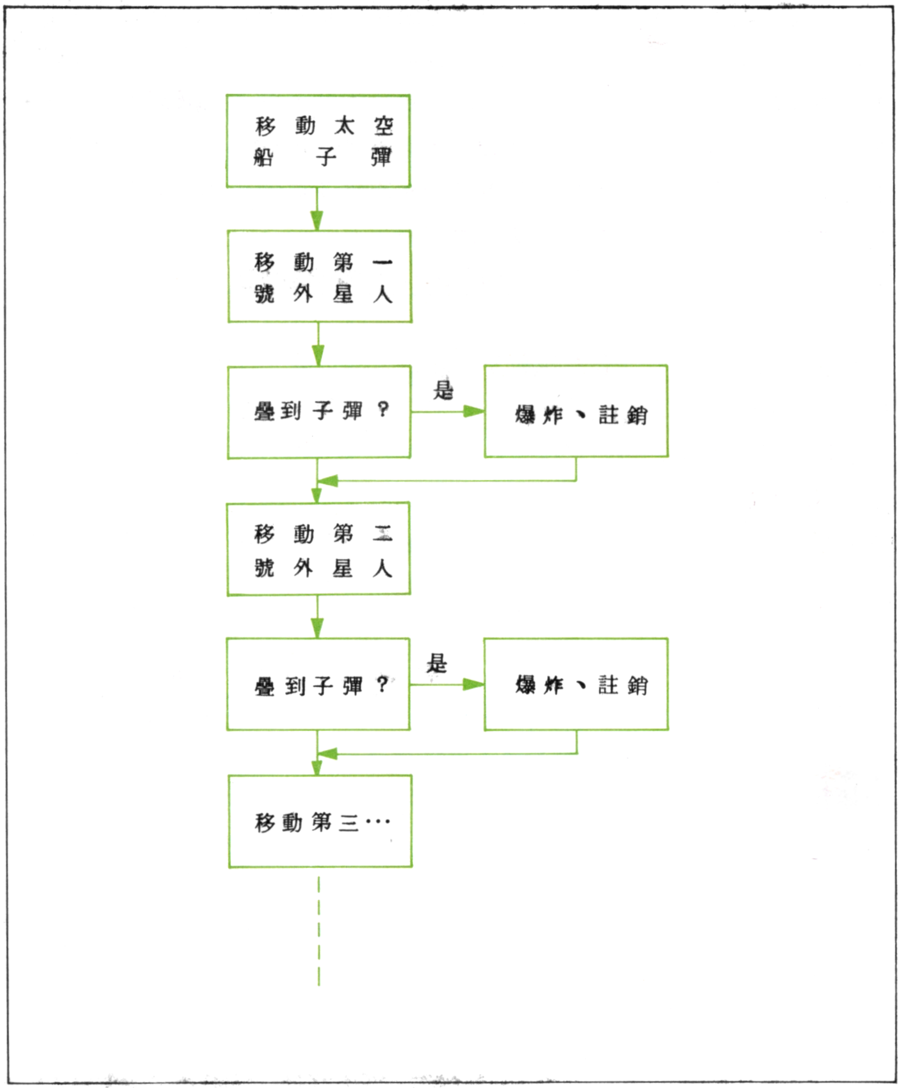

設定初始狀態
畫太空船及外星人
讀取搖桿值
太空船消圖
是否有子彈？
是
移動子彈
否
按鈕按下？
是
發射子彈
外星人消圖
在新位置畫外星人
外星人是否疊到子彈
否
是
該外星人註銷，加分爆炸，發聲，外星人消圖
移外星人子彈
畫太空船
太空船是否疊到子彈
否
是
失去一艘太空船
延時

圖 5 - 8　不良的順序流程

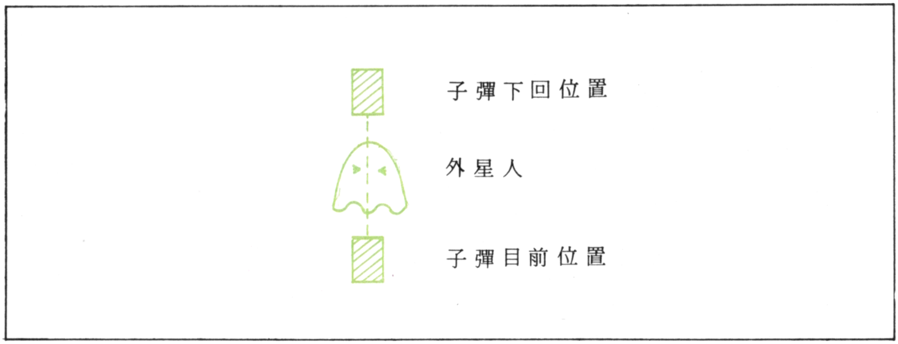
首先要指出的是圖 5 — 8 當中，太空船消圖太早，而外星人繪圖消圖時間很長，太空船會呈現閃爍現象。所以合理的安排是將太空船消圖的動作移到外星人子彈移動之前、外星人移動之後。另外一個嚴重的問題是當外星人數目增多時，外星人移動的動作佔用了大部份時間，因此使得太空船及雙方子彈移動顯得緩慢（作者曾試過30個 3 位元組寬、8 線高的外星人，此時子彈速度慢得不合理），所以這種方式只適用於一次發射一整束雷射線的場合（如大進擊II），而不適於小蜜蜂、長生鳥等發射子彈的場合。

那麼為什麼圖 5 — 8 的流程是這樣的安排呢？原來在某些設計者的觀點裡，子彈必須比物體先移動，也就是說當太空船的子彈向前移動一次後，整批外星人就必須一一移動一次，而當使用 DRAW 副程式（如程式 4 — 2 的 DRAW副程式）來畫外星人時，一旦外星人疊到（碰到）太空船的子彈，則該號外星人便要被摧毀了。所以圖 5 — 8 更詳細的動作部份是如下圖示：



子彈必須等外星人一一移動後才能再移動一次，自然是緩慢無比了。以這種結構寫出來的程式，我曾做過試驗，該程式執行時，最初子彈及太空船的

移動均很緩慢，而當多數外星人被打下之後，外星人移動時間減少（個數少了），結果畫面上子彈、太空船、外星人的動作均加快很多。這樣的設計，當外星人數少的時候，速度變快，或許有助於程式的戲劇效果，也常見於某些 Game裡，但是太空船發射子彈由最初很慢變成急快，無論如何也是不合適的吧！如果為了加快子彈速度而將子彈的移動間距加大，則會發生子彈越過外星人，而卻未擊中的情形。



那麼這樣的問題如何解決呢？檢視上述的說法，問題的關鍵是在外星人群的移動上，一個方法是，為了保持速度的一定，當外星人減少一個時，便代之以一個相等時間的延時廻路，這樣速度便固定了。可是這樣仍然沒有解決子彈速度緩慢的問題，解決的方法是，外星人不必全部一次畫完，只要每次一個外星人加入廻圈便可以了，而每走一回廻圈，便換下一個外星人，如此連續幾次廻圈以後，外星人便全部移動了，其畫面效果與原來一樣，但由於太空船與子彈每次均加入廻圈，所以子彈的速度便加快許多了（數十倍）。

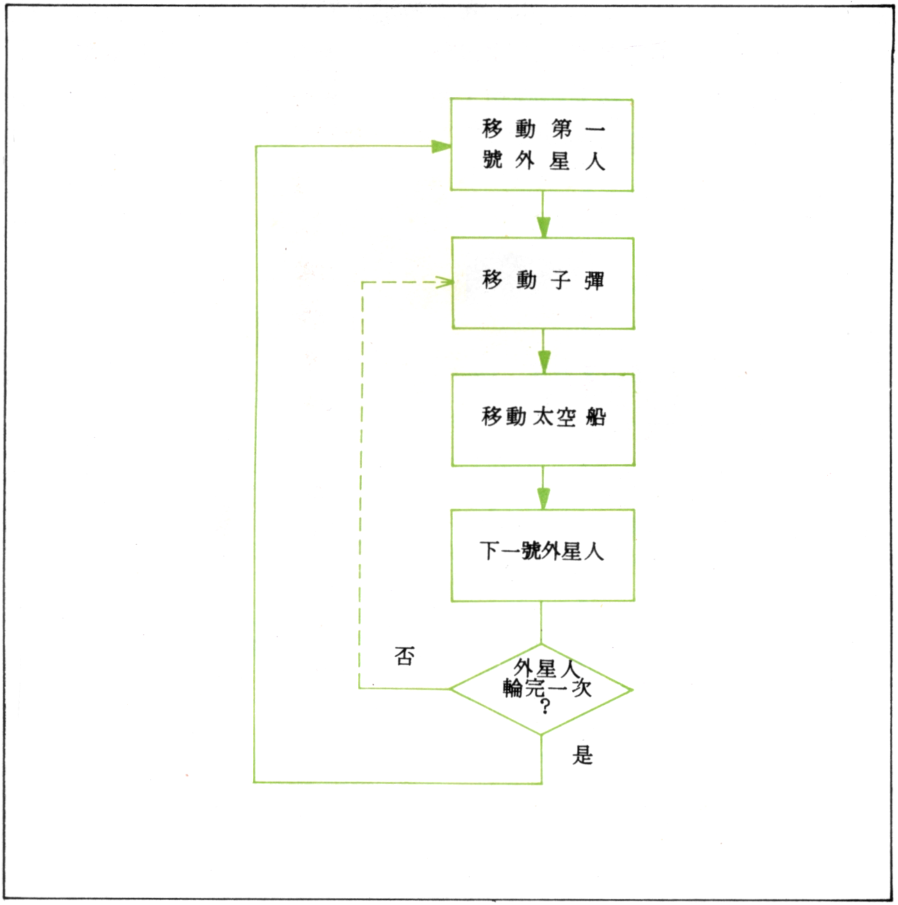
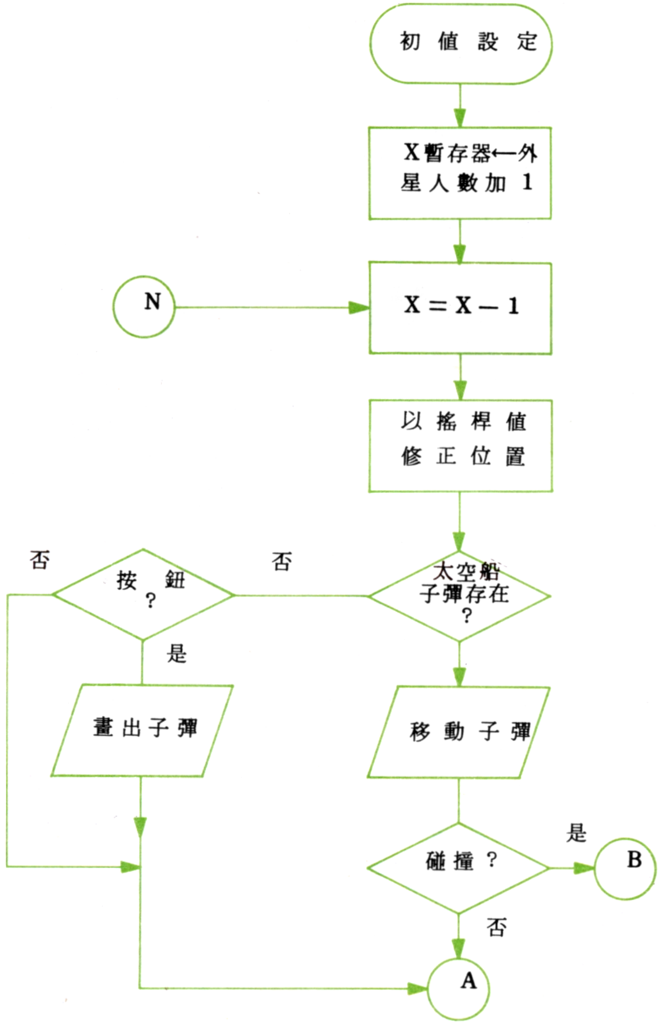


圖 5 - 9　正確的順序結構

反對這種做法的人，認為外星人必須在移動子彈之後畫圖，如此才有辦法知道是哪一號外星人與子彈碰撞，如果先畫了外星人再畫子彈，則子彈一旦撞上外星人，却無法直接檢知到是哪一號外星人被碰撞了。但事情也不如此悲觀，子彈一碰撞上物體之後，我們馬上可以令程式轉移到一個檢知副程式，而這個副程式只要將每個外星人在原位置重畫一遍，便可以

知道是在哪一號外星人位置上有一顆子彈了！可是每一號外星人都重畫過，速度是不是又慢了？其實一旦有碰撞發生，接下來的便是發聲與爆炸了，而這些動作皆佔用了很長的時間，相對的來說全部外星人的畫圖時間便不會造出任何明顯的延時了。


初 值 設 定
X暫存器←外星人數加 1
N
X＝X－1
以搖桿值修正位置
否
按 鈕 ?
否
太空船子彈存在 ?
是
畫出子彈
移動子彈
是
碰撞 ?
B
否
A

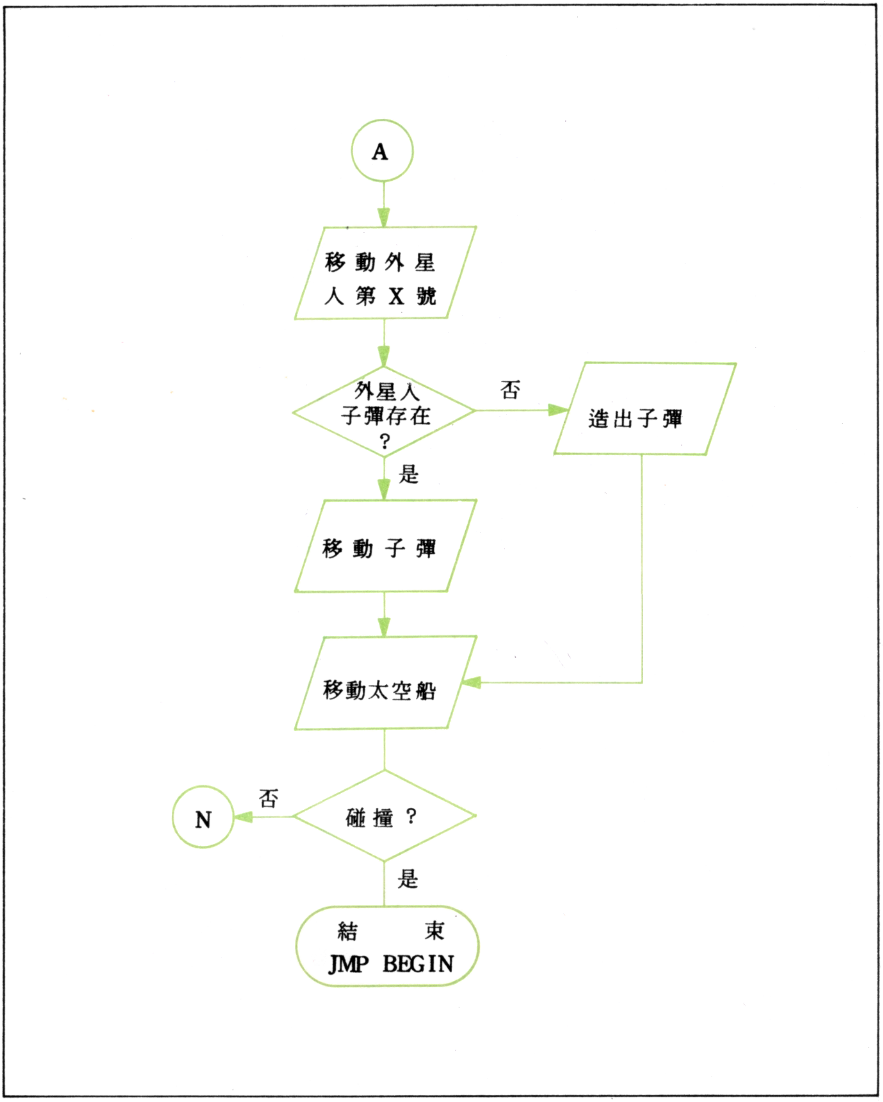


圖 5 -10　　程式 5 - 6 的主流程

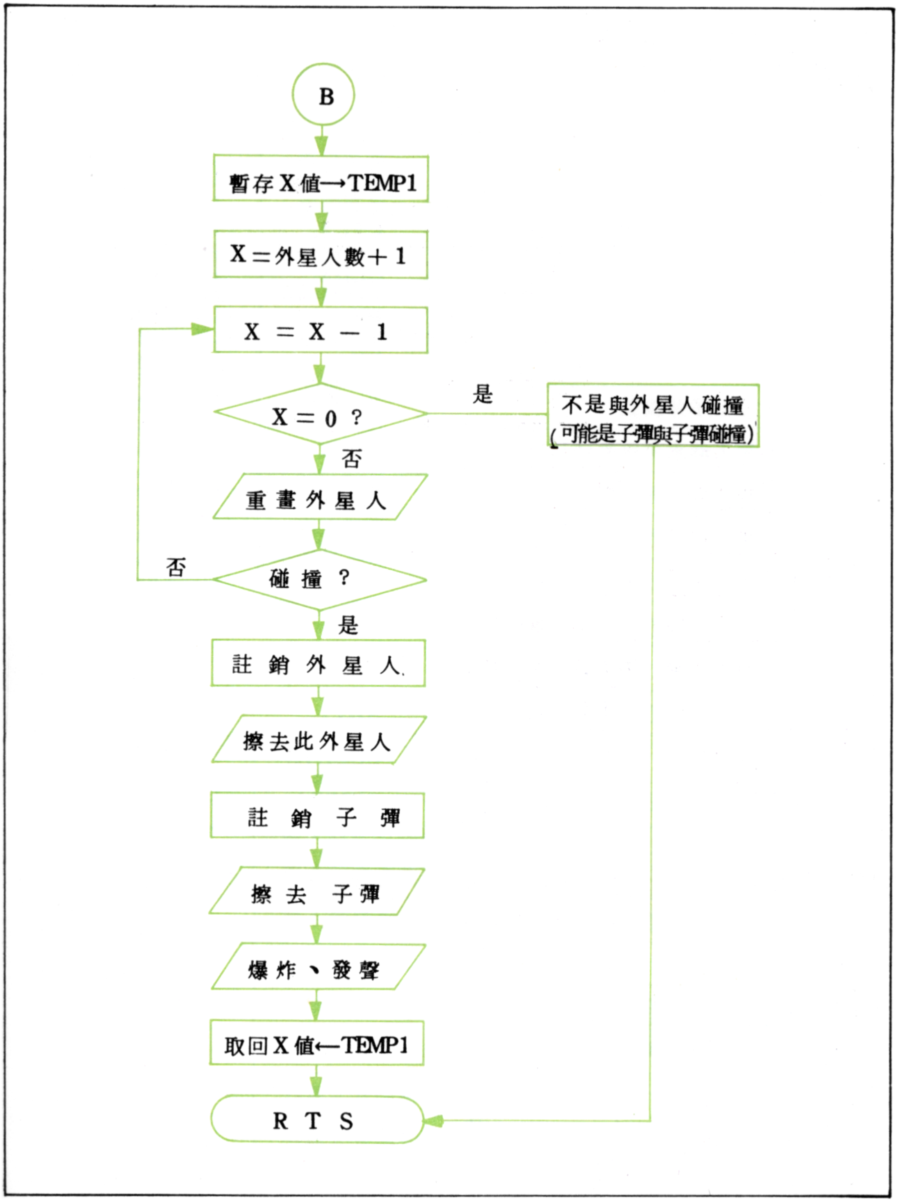


圖 5 - 11　碰撞檢知副程式

程式5-6的變數提要

功　　　能	名　稱	0頁位址
太空船水平位置	CTLX	$F6
太空船先前水平位置	OLDX	$F7
太空船子彈水平位置	BX	$F1
太空船子彈垂直位置	BY	$F2
太空船子彈旗號	FLAG	$F3
外星人群水平位置陣列位址	NEWX, X	$5000, X
外星人群垂直位置陣列位址	NEWY, X	$5020, X
外星人群生存旗號陣列位址	KILLED, X	$5040, X
外星人子彈（第0號）水平位址	ABX	$E6
外星人子彈（第0號）垂直位置	ABY	$E7
外星人子彈（第1號）水平位置	ABX1	$E8
外星人子彈（第1號）垂直位置	ABY1	$E9
碰撞旗號	COLLIS	$0B
圖形號碼指位器	SHNUM	$03

*SHNUM＝0：外星人，1：外星人子彈，2：太空船，
3：太空船子彈，4：爆炸火花。

副程式及進入點提要

名　稱	功　　　能
BEGIN	程式開始，設定畫面、旗號、變數初始值
READPDL	讀取搖桿值，以修定目前太空船水平位置變數

BULLET	移動或新造太空船子彈
BNEW	新造太空船子彈
MOVE	移動太空船子彈
DETECT	碰撞檢知
ALIEN	移動外星人
ABULLET	移動第 0 號外星人子彈
ABULLET1	移動第 1 號外星人子彈
XDRSHIP	移動太空船
DETECT1	太空船是否與子彈碰撞
SIDE	改變速度（VX）方向副程式
XCLEAR	爆炸及發聲副程式
FIND	檢知被擊中外星人號碼的副程式
DRAW	畫圖（EOR）副程式

位址表

位址表名稱	功能
SHTAB	圖序表起始位址表
SHTAB0	第 0 號圖序表（外星人）
SHTAB1	第 1 號圖序表（外星人子彈群）
SHTAB2	第 2 號圖序表（太空船）
SHTAB3	第 3 號圖序表（太空船子彈）
SHTAB4	第 4 號圖序表（爆炸圖形）
LOCX	外星人畫面初始水平位置座標
LOCY	外星人畫面初始垂直位置座標

程式說明

本程式可以由圖 5 —10的流程及程式本身的附註來瞭解，不過有些地方仍有說明的必要：程式由第61行 BEGIN 開始，然後由 NEXT 廻圈畫出10個外星人，第90行的 DONE 用來等候按鍵，按鍵之後便畫出太空船，然後進行遊戲的大廻圈。

第107 行便是輪流移動外星人廻圈的起點，第 114 行～第145 行則讀取搖桿值來修正太空船水平位置，改變號 125 行及 141 行的 SBC及ADC值（此處爲 #$04 ），可以改變太空船水平移動的速度，不過如果採用了彩色圖形，則要保持偶數位移。

152 行至 183 行是用來移動太空船的子彈，158 行的LDX OLDX 便是以太空船位置來當作發射點，但是我們的圖形位置是以圖形左上頂點爲準的，因此必如156 行那樣加上 #$03 才能令子彈由太空船中央發射。（但當太空船移到最右邊時，子彈却由畫面最左邊發射，怎麼辦？）。有些程式（如長生鳥）由於沒有加入這個修正步驟，而使得子彈竟然由太空船的左方發射（所以，如果您是行家，只要看畫面動作便可知道程式的動作內容了。）。184 ～ 187 是子彈發聲程式 *，192 行開始是碰撞檢查，一旦碰撞發生，便要到331 行的 FIND副程式去找出被撞者的號碼了。 229 行～ 283 行是外星人子彈發射及移動處理，其構造類似於152 行的 BULLET，但是此處並不要有按鈕才發射，只要畫面上沒有子彈便造出新的子彈來，而其發射位置是由 233 行及 259 行以 LDA OLDX 來瞄準太空船的。改變 246 行及 273行ADC的值（此處爲 #$01 ）可以改變子彈速度（可否改得更慢？），一旦子彈超過畫面底端（$B0 ）便被消去，並且註銷其存在（ 249 ～ 251 行）。

288 行～ 306行便是移動太空船的程式，如果太空船被擊中，便爆炸、發聲並且遊戲重新開始。 311 行～ 316行（ SIDE ）是改變外星人移動

* 見附錄 C

方向的副程式，如果正方向速度爲 #$02，則反方向速度是 #$FE（譬如 #$08 + #$FE = #$06，C = 1），SIDE副程式便是以此方法來處理的。XCLEAR是爆炸發聲副程式，其中 SOUND = $FBDD 是一個長爲 0.5 秒的發聲程式，因此便夾在兩個 JSR DRAW 當中，兼有延時作用。331 行～365 行的FIND就如圖 5 —11的流程一樣，其中要注意的是，如果太空船子彈與外星人子彈相撞，也是會到FIND副程式裡來，因此 337 行～340 行便是沒有外星人被擊中的處理情況，其中 TEMP2 便是外星人被殺的旗號，如果有外星人被殺則 TEMP2 ≠ 0，因此MAX減一，否則總數不減。您也可以在 JSR FIND 之後，且 TEMP 2 = 0 的場合，將程式轉移（JSR）到子彈相撞副程式加以處理（銀河旅行便是）。368 ～ 498 便是我們的核心畫圖工具 DRAW副程式，相信大家已很熟悉，也就不多做說明了。

怎樣寫 Arcade Game

- 我們有一個DRAW ROUTINE
- 我們有四種繪圖法
- 我們有許多種顏色
- 我們有自動塗色程式
- 我們有自動造表程式
- 您有比我更豐富的創造力

我們便以這些工具來造出令人不可思議的動畫程式!!

下一步？

對於學習而言，一個雛形的範例是比複雜的程式有用得多，並且只要再加修改，你便不難造出一個商用程式來，譬如加入特殊音效（參考R 4）、增加遊戲難度、多重場景（幕）變換…… 等等，只待配合您的創作力

，一切便很完美了。並且，如果你眞正的自行設計過第一個程式，接下來的便簡單得多。而您也會同意小精靈之類的 Game 也不是怎樣困難的事了。

有關Arcade Game 我們要說的，便是這些！現在請進入 Advanture Game 的領域………。

小蜜蜂程式舉例

```
!L
 ;
 *********************************
 * APPLE INVADERS SAMPLE PROGRAM *
 *********************************
 *
 * 0-PAGE ADDRESS
 *
 XBYTE     EPZ $00
 XBIT      EPZ $01
 VERT      EPZ $02
 SHNUM     EPZ $03
 SCX       EPZ $04
 HIRESL    EPZ $06
 HIRESH    EPZ $07
 WIDTH     EPZ $08
 HEIGHT    EPZ $09
 XCOUNT    EPZ $0A
 COLLIS    EPZ $0B
 ABX       EPZ $E6
 ABY       EPZ $E7
 ABX1      EPZ $E8
 ABY1      EPZ $E9
 AFLAG     EPZ $EA
 AFLAG1    EPZ $EB
 TEMP1     EPZ $EC
 TEMP2     EPZ $ED
 TEMP3     EPZ $EE
 TEMP4     EPZ $EF
 OLD       EPZ $F0
 BX        EPZ $F1
 BY        EPZ $F2
```

```
FLAG      EPZ $F3
MAX       EPZ $F5
PDL       EPZ $F4
CTLX      EPZ $F6
OLDX      EPZ $F7
VX        EPZ $F8
A         EPZ $F9
Y         EPZ $FA
X         EPZ $FB
TEMP      EPZ $FF
*
* NON 0-PAGE ADDRESS
*
NEWX      EQU $5000
NEWY      EQU $5020
KILLED    EQU $5040
KEY       EQU $C000
RESETKEY  EQU $C010
SPEAKER   EQU $C030
DELAY     EQU $FCA8
PREAD     EQU $FB1E
BUTTON    EQU $C061
SOUND     EQU $FBDD
*===============================*
          ORG $A00
*===============================*
*
* BEGIN SET UP
*
BEGIN     JSR $F3E2     ; HGR
          BIT $C052     ; 全畫面
          LDA #$00
          STA FLAG
          STA AFLAG
          STA AFLAG1
          STA SHNUM
          LDY #$0A      ; 10 個外星人
          STY MAX
          LDA #$02      ; 速度 = 2
          STA VX
          LDA #LOCX     ; (TEMP1 , TEMP2) 指向 LOCX 位址
          STA TEMP1
          LDA /LOCX
          STA TEMP2
```

```
              LDA  #LOCY
              STA  TEMP3            ; (TEMP3,TEMP4) 指
              LDA  /LOCY            ;  向LOCY位址
              STA  TEMP4
     NEXT     LDA  (TEMP1),Y        ; 設定(NEWX,NEWY)←(
              STA  NEWX,Y           ;  LOCX,LOCY)
              STA  SCX
              LDA  (TEMP3),Y
              STA  NEWY,Y
              STA  VERT
              JSR  DRAW
              LDA  #$00
              STA  KILLED,Y         ; KILLED,Y = 0
              DEY
              BNE  NEXT
     DONE     LDA  KEY              ; 等候按鍵
              BPL  DONE
              STA  RESETKEY
              LDA  #$02             ; 畫太空船
              STA  SHNUM
              LDA  #$A0
              STA  VERT
              LDA  #$80
              STA  SCX
              STA  CTLX
              STA  OLDX
              JSR  DRAW
     ********************************
     * MAIN PROGRAM BEGIN HERE  !!! *
     ********************************
     *
     LOADMAX  LDX  #$0B             ; X←外星人數+1
     NEXTMAX  DEX                   ; X=X-1
              BEQ  LOADMAX          ; 當輪完一回則重頭開始
     ************************
     * READ PADDLE SECTION *
     ************************
     *
     READPDL  STX  TEMP1            ; 暫存X值
              LDX  #$01             ; 讀PDL(1)值
     *        STX  OLD
              JSR  PREAD
              LDX  TEMP1            ; 取回X值
              STY  PDL              ; 其它PDL說明見圖3-4
```

```
               TYA
               CMP  CTLX
               BGE  PDL3
               LDA  CTLX
               SEC
               SBC  #$04
               BGE  PDL1
               LDA  #$00
               STA  CTLX
PDL1           CMP  PDL
               BGE  PDL5
               LDA  PDL
PDL2           STA  CTLX
               JMP  PDL5
PDL3           CMP  CTLX
               BEQ  PDL4
               LDA  CTLX
               CMP  #$F8
               BGE  PDL4
               LDA  CTLX
               CLC
               ADC  #$04
               CMP  PDL
               BLT  PDL5
PDL4           LDA  PDL
PDL5           STA  CTLX
               JMP  BULLET
*
*
*************************
*    NOW MOVE BULLET    *
*************************
BULLET         LDA  #$03       ; 太空船子彈，SHNUM= 3
               STA  SHNUM
               LDA  FLAG       ; 子彈FLAG≠0？(存在)
               BNE  MOVE
               LDA  BUTTON     ; 按鈕？
               BPL  ALIEN
BNEW           LDA  OLDX       ; 取太空船位置
               CLC
               ADC  #$03       ; 調整至船中央位置
               STA  BX
               STA  SCX
               LDA  #$9A
```

```
                STA  BY
                STA  VERT
                STA  FLAG             ；設定FLAG≠0
                JMP  NMOVE
MOVE            LDA  BX
                STA  SCX
                LDA  BY
                STA  VERT
                SEC
                SBC  #$08             ；向上移動 8 格
                STA  BY
                BGE  CONTIO           ；若超出螢幕上方（BY<0）
                LDA  #$00             ；則消去子彈
                STA  FLAG
                JSR  DRAW
                JMP  ALIEN
CONTIO          JSR  DRAW
                LDA  BY
                STA  VERT
NMOVE           JSR  DRAW
                LDY  #$01
SOUNDO          STA  SPEAKER          ；觸動喇叭
                DEY
                BNE  SOUNDO
**********************
*   DETECT COLLISION  *
**********************
*
DETECT          LDA  COLLIS           ；是否碰撞？
                BEQ  ALIEN
                JSR  FIND             ；是，則至FIND
                LDA  TEMP2            ；是否撞到外星人？
                BEQ  ALIEN            ；否，則JMP到ALIEN
                DEC  MAX              ；外星人數減一
                BNE  ALIEN
                JMP  BEGIN
**************************
*    NOW MOVE ALIEN !!!   *
**************************
*
ALIEN           LDA  #$00             ；外星人，SHNUM＝ 0
                STA  SHNUM
                LDA  KILLED,X         ；這個外星人是否已被殺？
                BEQ  ALIEN1
```

```
             JMP ABULLET      ; 若是，則跡過
ALIEN1       LDA NEWX,X
             STA SCX
             LDA NEWY,X
             STA VERT
             JSR DRAW
CONTI1       LDA NEWX,X
             CLC
             ADC VX           ; 以VX的速度移動
             CMP #$02         ; 是否到達邊界？
             BCS CONO
             JMP SIDE         ; 是，則到SIDE
CONO         STA NEWX,X
             STA SCX
CON          JSR DRAW
             LDA #$20         ; 延時
             JSR DELAY
******************************
*    NOW MOVE ALIEN BULLETS    *
******************************
*
ABULLET      LDA #$01         ; 外星人子彈，SHNUM＝1
             STA SHNUM
             LDA AFLAG        ; 子彈第 0 號是否已存在
             BNE MOVEAB
ABNEW        LDA OLDX         ; 以太空船目前位置 OLDX
             STA ABX          ; 來瞄準，並創造子彈
             STA SCX
             LDA #$20
             STA ABY
             STA AFLAG        ; AFLAG ≠ 0
             STA VERT
             JMP XAB
MOVEAB       LDA ABX
             STA SCX
             LDA ABY
             STA VERT
             CLC
             ADC #$01         ; 下移一格
             CMP #$B0         ; 若超出螢幕則註銷之
             BLT EXIST
             LDA #$00
             STA AFLAG
             JMP XAB
```

```
EXIST      STA  ABY
           JSR  DRAW
           LDA  ABY
           STA  VERT
XAB        JSR  DRAW
ABULLET1   LDA  AFLAG1        ；子彈第 1 號，說明同第 0 號
           BNE  MOVEAB1
ABNEW1     LDA  OLDX
           ADC  #$03
           STA  ABX1
           STA  SCX
           LDA  #$2A
           STA  AFLAG1
           STA  ABY1
           STA  VERT
           JMP  XAB1
MOVEAB1    LDA  ABX1
           STA  SCX
           LDA  ABY1
           STA  VERT
           CLC
           ADC  #$01
           CMP  #$B0
           BLT  EXIST1
           LDA  #$00
           STA  AFLAG1
           JMP  XAB1
EXIST1     STA  ABY1
           JSR  DRAW
           LDA  ABY1
           STA  VERT
XAB1       JSR  DRAW
**********************
*    NOW MOVE SHIP   *
**********************
*
XDRSHIP    NOP                ；太空船，SHNUM＝2
CONTI2     LDA  #$02
           STA  SHNUM
           LDA  OLDX          ；取舊位置
           STA  SCX
           LDA  #$A0
           STA  VERT
           JSR  DRAW          ；消圖
```

```
DRSHIP    LDA CTLX     ；取新位置
          STA SCX
          STA OLDX
          LDA #$A0
          STA VERT
          JSR DRAW     ；畫圖
DETECT1   LDA COLLIS   ；是否碰撞
          BEQ CONTI3
          JSR XCLEAR   ；若是則 JSR 到 XCLEAR
          JMP BEGIN    ；結束遊戲
CONTI3    JMP NEXTMAX  ；若否，則繼續進行廻圈
***************************
* CHANGE DIRECTION ROUTINE *
***************************
*
SIDE      LDA VX       ；取 V X
          EOR #$FF     ；EOR，取補數
          CLC
          ADC #$01     ；V X 取 2 的補數＝－V X
          STA VX
          JMP CON
***************************
* EXPLODE & SOUND ROUTINE *
***************************
*
XCLEAR    LDA #$04     ；爆炸，SHNUM＝4
          STA SHNUM
          JSR DRAW     ；畫爆炸圖
          JSR SOUND    ；發聲延時
          JSR DRAW     ；消爆炸圖
          RTS
*********************************
* SEARCH WHICH ONE WAS COLLIDED *
*********************************
*
FIND      LDA #$00     ；外星人，SHNUM＝0
          STA SHNUM
          STX TEMP1    ；TEMP1←X
          LDX #$0B     ；X＝外星人數＋1
XXNEXT    DEX          ；X＝X－1
          BNE XXCONTI  ；全部找過？否，則至 XXCONTI
          LDA #$00
          STA TEMP2    ；是，TEMP2←0
          LDX TEMP1    ；TEMP1 →X
```

```
                  RTS                ; 結束
XXCONTI           LDA KILLED,X       ; 這個外星人存在？
                  BNE XXNEXT         ; 否，下一號
                  LDA NEWX,X
                  STA SCX
                  LDA NEWY,X
                  STA VERT
                  JSR DRAW           ; 原地畫圖消圖
                  JSR DRAW
                  LDA COLLIS         ; 碰損？
                  BEQ XXNEXT         ; 否，下一個
                  STA KILLED,X       ; 是，消去外星人
                  JSR DRAW
                  LDA #$03           ; 消去子彈
                  STA SHNUM
                  STA TEMP2          ; 設定 TEMP2，外星人被殺
                  LDA BX
                  STA SCX
                  LDA BY
                  STA VERT
                  JSR DRAW
                  LDA #$00
                  STA FLAG           ; 註銷子彈
                  JSR XCLEAR         ; 爆炸發聲
                  LDX TEMP1          ; TEMP1←X
                  RTS
*=========END OF MAIN PROGRAM !!!
*
*****************************
* PROCEDURE DRAW (SCX,VERT) *
* LOCAL VARIABLES: VERT1,   *
* (SHLO,SHHI)               *
*****************************
*
VERT1             EPZ $0C
SHLO              EPZ $0D
SHHI              EPZ $0E
*
* NON-0-PAGE ADDRESS
*
TABHI             EQU $5100
TABLO             EQU $51C0
*****************************
* MAIN ROUTINE BEGIN HERE ! *
*****************************
```

```
*
DRAW      STA A          ;說明類似程式 4-0，4-2
          STY Y
          STX X
          LDY #$00
          LDA SCX
          CLC
*
* DIVIDE BY '7' ROUTINE
*
DLOOP     CMP #$07
          BCC DDONE
          SEC
          SBC #$07
          INY
          CLC
          JMP DLOOP
DDONE     STA XBIT
          STY XBYTE
*
* SET UP COLLIS FLAG
* AND SHAPE NUMBER
*
          LDA #$00
          STA COLLIS
          LDA SHNUM
          ASL
          TAY
          LDA SHTAB,Y
          STA SHLO
          INY
          LDA SHTAB,Y
          STA SHHI
*
* (SHLO,SHHI) IS NOW POINT TO
* THE CORRESPONDING SHAPE ADDRESS
*
*
* NOW SET UP WIDTH AND HEIGHT
*
          LDY #$0E
          LDA (SHLO),Y
          STA WIDTH
          INY
          LDA (SHLO),Y
```

```
                STA HEIGHT
*
* GET SHTAB ADDRESS
*
                LDA XBIT
                ASL
                TAY
                LDA (SHLO),Y
                TAX
                INY
                LDA (SHLO),Y
*
* AUTO-MODIFY P.S.#
*
                CLC
                ADC SHLO
                STA COLL+1
                TXA
                ADC SHHI
                STA COLL+2
                LDA VERT
                STA VERT1
*
* BEGIN DRAWING ONE SHAPE
*
                LDX #$02
LOOP1           LDA WIDTH
                STA XCOUNT
                LDY VERT1
                CPY #$C0
                BCC YTABLE
                LDY #$29
                BCS LOOP2
*
* LOOKING UP SCREEN ADDR TABLE
*
YTABLE          LDA TABLO,Y
                STA HIRESL
                LDA TABHI,Y
                STA HIRESH
                LDY XBYTE
*
* DRAW A LINE
*
```

```
LOOP2      CPY #$28
           BCS CONT
* -----------------SET COLLIS FLAG
COLL       LDA COLL,X
           STA TEMP
           AND (HIRESL),Y
           AND #$7F
           BEQ BLOCK1
           STA COLLIS
* ----------- STORE DATA ON SCREEN
BLOCK1     LDA TEMP
           EOR (HIRESL),Y
           STA (HIRESL),Y
CONT       INX
           INY
           DEC XCOUNT
           BNE LOOP2
* ----------------END OF DRAW LINE
           INC VERT1
           DEC HEIGHT
           BNE LOOP1
           LDA A
           LDY Y
           LDX X
           RTS
* ---------------END OF DRAW SHAPE
SHTAB      ADR SHTAB0              ; 外星人圖序表起始位址
           ADR SHTAB1              ; 外星人子彈圖序表起始位址
           ADR SHTAB2              ; 太空船圖序表起始位址
           ADR SHTAB3              ; 太空船子彈圖序表起始位址
           ADR SHTAB4              ; 爆炸圖序表起始位址
SHTAB0     HEX 000E003A00660092
           HEX 00BE00EA0116030E
           HEX 0000000000004002
           HEX 00500A0014280014
           HEX 2800142800142800
           HEX 1428001428001428
           HEX 00142800500A0040
           HEX 0200
           HEX 030E000000000000
           HEX 0005002015002855
           HEX 0008450028550028
           HEX 5500084500084000
           HEX 0840000840002015
```

```
               HEX 00000500
               HEX 030E000000000000
               HEX 000A00402A00
               HEX 502A01502A015020
               HEX 0150200150200150
               HEX 2001502001100001
               HEX 402A00000A00
               HEX 030E0000
               HEX 0000000000140000
               HEX 5500205502201402
               HEX 2055022055022041
               HEX 0220410220000220
               HEX 0002005500001400
               HEX 030E
               HEX 0000000000000028
               HEX 00002A01402A0540
               HEX 2804402A05402A05
               HEX 402A054002054000
               HEX 04400004002A0100
               HEX 2800
               HEX 030E000000000000
               HEX 0050000054020055
               HEX 0A00550A00050A00
               HEX 050A00050A00050A
               HEX 00050A0001080054
               HEX 02005000
               HEX 030E00000000
               HEX 0000002001002805
               HEX 002A15002211002A
               HEX 15002A1500221100
               HEX 2211000210000210
               HEX 002805002001
      SHTAB1   HEX 000E002000320044
               HEX 00560068007A0208
               HEX 1201120154005400
               HEX 10002A0102010000
               HEX 0208080008000800
               HEX 080008003E001C00
               HEX 08000208A885D482
               HEX A885D482A8852805
               HEX 500228050208A885
               HEX D082A08140014001
               HEX 4001400140010208
               HEX 1002200140004000
```

```
       HEX 4000200120010000
       HEX 02084007600F600F
       HEX 4007000000000000
       HEX 00000208808AC0AA
       HEX C0AA008A00000000
       HEX 00000000
SHTAB2 HEX 000E001C002A0038
       HEX 0046005400620206
       HEX 1000100054005601
       HEX 02017E0102062000
       HEX 200028012C030402
       HEX 7C03020640004000
       HEX 5002580608047807
       HEX 0206000100012005
       HEX 300D1008700F0206
       HEX 00020002400A601A
       HEX 2010601F02060004
       HEX 0004001540354020
       HEX 403F020600080008
       HEX 002A006B0041007F
SHTAB3 HEX 000E00180022002C
       HEX 00360040004A0204
       HEX 0300030003000300
       HEX 0204060006000600
       HEX 060002040C000C00
       HEX 0C000C0002041800
       HEX 1800180018000204
       HEX 3000300030003000
       HEX 0204600060006000
       HEX 6000020440014001
       HEX 40014001
SHTAB4 HEX 000E00280042005C
       HEX 0076009000AA0308
       HEX 2000002204002204
       HEX 0028010028010020
       HEX 0000220400220400
       HEX 0308400000440800
       HEX 4408005002005002
       HEX 0040000044080044
       HEX 0800030800010008
       HEX 1100081100200500
       HEX 2005000001000811
       HEX 0008110003080002
       HEX 0010220010220040
       HEX 0A00400A00000200
```

```
                HEX 1022001022000308
                HEX 0004002044002044
                HEX 0000150000150000
                HEX 0400204400204400
                HEX 0308000800400801
                HEX 400801002A00002A
                HEX 0000080040080140
                HEX 0801030800100000
                HEX 1102001102005400
                HEX 0054000010000011
                HEX 02001102
     LOCX       HEX 008565A54585C525   ;外星人在畫面初
                HEX 65A5E5              始位置座標值
     LOCY       HEX 00020C0C16161620
                HEX 202020
                END
```

程式 5 - 6　小蜜蜂式的 Game 舉例

* 本程式不能在 64K LISA 執行，若爲 64K 系統，請由 DOS BRUN。

```
!BR
*A00.104C

0A00- 20 E2 F3 2C 52 C0 A9 00
0A08- 85 F3 85 EA 85 EB 85 03
0A10- A0 0A 84 F5 A9 02 85 F8
0A18- A9 35 85 EC A9 10 85 ED
0A20- A9 40 85 EE A9 10 85 EF
0A28- B1 EC 99 00 50 85 04 B1
0A30- EE 99 20 50 85 02 20 3C
0A38- 0C A9 00 99 40 50 88 D0
0A40- E7 AD 00 C0 10 FB 8D 10
0A48- C0 A9 02 85 03 A9 A0 85
0A50- 02 A9 80 85 04 85 F6 85
0A58- F7 20 3C 0C A2 0B CA F0
0A60- FB 86 EC A2 01 86 F0 20
0A68- 1E FB A6 EC 84 F4 98 C5
0A70- F6 B0 16 A5 F6 38 E9 04
0A78- B0 04 A9 00 85 F6 C5 F4
```

```
0A80-  B0  1C  A5  F4  85  F6  4C  9E
0A88-  0A  C5  F6  F0  16  A5  F6  C9
0A90-  F8  B0  09  A5  F6  18  69  04
0A98-  C5  F4  90  02  A5  F4  85  F6
0AA0-  4C  A7  0A  A9  00  85  F0  A9
0AA8-  03  85  03  A5  F3  D0  19  AD
0AB0-  61  C0  10  51  A5  F7  18  69
0AB8-  03  85  F1  85  04  A9  9A  85
0AC0-  F2  85  02  85  F3  4C  E8  0A
0AC8-  A5  F1  85  04  A5  F2  85  02
0AD0-  38  E9  08  85  F2  B0  0A  A9
0AD8-  00  85  F3  20  3C  0C  4C  05
0AE0-  0B  20  3C  0C  A5  F2  85  02
0AE8-  20  3C  0C  A0  01  8D  30  C0
0AF0-  88  D0  FA  A5  0B  F0  0E  20
0AF8-  F0  0B  A5  ED  F0  07  C6  F5
0B00-  D0  03  4C  00  0A  A9  00  85
0B08-  03  BD  40  50  F0  03  4C  38
0B10-  0B  BD  00  50  85  04  BD  20
0B18-  50  85  02  20  3C  0C  BD  00
0B20-  50  18  65  F8  C9  02  B0  03
0B28-  4C  D6  0B  9D  00  50  85  04
0B30-  20  3C  0C  A9  20  20  A8  FC
0B38-  A9  01  85  03  A5  EA  D0  11
0B40-  A5  F7  85  E6  85  04  A9  20
0B48-  85  E7  85  EA  85  02  4C  70
0B50-  0B  A5  E6  85  04  A5  E7  85
0B58-  02  18  69  01  C9  B0  90  07
0B60-  A9  00  85  EA  4C  70  0B  85
0B68-  E7  20  3C  0C  A5  E7  85  02
0B70-  20  3C  0C  A5  EB  D0  13  A5
0B78-  F7  69  03  85  E8  85  04  A9
0B80-  2A  85  EB  85  E9  85  02  4C
0B88-  A9  0B  A5  E8  85  04  A5  E9
0B90-  85  02  18  69  01  C9  B0  90
0B98-  07  A9  00  85  EB  4C  A9  0B
0BA0-  85  E9  20  3C  0C  A5  E9  85
0BA8-  02  20  3C  0C  EA  A9  02  85
0BB0-  03  A5  F7  85  04  A9  A0  85
0BB8-  02  20  3C  0C  A5  F6  85  04
0BC0-  85  F7  A9  A0  85  02  20  3C
0BC8-  0C  A5  0B  F0  06  20  E2  0B
0BD0-  4C  00  0A  4C  5E  0A  A5  F8
0BD8-  49  FF  18  69  01  85  F8  4C
```

```
0BE0-  30 0B A9 04 85 03 20 3C
0BE8-  0C 20 DD FB 20 3C 0C 60
0BF0-  A9 00 85 03 86 EC A2 0B
0BF8-  CA D0 07 A9 00 85 ED A6
0C00-  EC 60 BD 40 50 D0 F1 BD
0C08-  00 50 85 04 BD 20 50 85
0C10-  02 20 3C 0C 20 3C 0C A5
0C18-  0B F0 DD 9D 40 50 20 3C
0C20-  0C A9 03 85 03 85 ED A5
0C28-  F1 85 04 A5 F2 85 02 20
0C30-  3C 0C A9 00 85 F3 20 E2
0C38-  0B A6 EC 60 85 F9 84 FA
0C40-  86 FB A0 00 A5 04 18 C9
0C48-  07 90 08 38 E9 07 C8 18
0C50-  4C 47 0C 85 01 84 00 A9
0C58-  00 85 0B A5 03 0A A8 B9
0C60-  D5 0C 85 0D C8 B9 D5 0C
0C68-  85 0E A0 0E B1 0D 85 08
0C70-  C8 B1 0D 85 09 A5 01 0A
0C78-  A8 B1 0D AA C8 B1 0D 18
0C80-  65 0D 8D B0 0C 8A 65 0E
0C88-  8D B1 0C A5 02 85 0C A2
0C90-  02 A5 08 85 0A A4 0C C0
0C98-  C0 90 04 A0 29 B0 0C B9
0CA0-  C0 51 85 06 B9 00 51 85
0CA8-  07 A4 00 C0 28 B0 13 BD
0CB0-  AF 0C 85 FF 31 06 29 7F
0CB8-  F0 02 85 0B A5 FF 51 06
0CC0-  91 06 E8 C8 C6 0A D0 E3
0CC8-  E6 0C C6 09 D0 C3 A5 F9
0CD0-  A4 FA A6 FB 60 DF 0C 21       ; SHTAB 由此
0CD8-  0E AD 0E 1D 0F 71 0F 00       ; 開始
0CE0-  0E 00 3A 00 66 00 92 00
0CE8-  BE 00 EA 01 16 03 0E 00
0CF0-  00 00 00 00 00 40 02 00
0CF8-  50 0A 00 14 28 00 14 28
0D00-  00 14 28 00 14 28 00 14
0D08-  28 00 14 28 00 14 28 00
0D10-  14 28 00 50 0A 00 40 02
0D18-  00 03 0E 00 00 00 00 00
0D20-  00 00 05 00 20 15 00 28
0D28-  55 00 08 45 00 28 55 00
0D30-  28 55 00 08 45 00 08 40
0D38-  00 08 40 00 08 40 00 20
```

```
0D40- 15 00 00 05 00 03 0E 00
0D48- 00 00 00 00 00 00 0A 00
0D50- 40 2A 00 50 2A 01 50 2A
0D58- 01 50 20 01 50 20 01 50
0D60- 20 01 50 20 01 50 20 01
0D68- 10 00 01 40 2A 00 00 0A
0D70- 00 03 0E 00 00 00 00 00
0D78- 00 00 14 00 00 55 00 20
0D80- 55 02 20 14 02 20 55 02
0D88- 20 55 02 20 41 02 20 41
0D90- 02 20 00 02 20 00 02 00
0D98- 55 00 00 14 00 03 0E 00
0DA0- 00 00 00 00 00 00 28 00
0DA8- 00 2A 01 40 2A 05 40 28
0DB0- 04 40 2A 05 40 2A 05 40
0DB8- 2A 05 40 02 05 40 00 04
0DC0- 40 00 04 00 2A 01 00 28
0DC8- 00 03 0E 00 00 00 00 00
0DD0- 00 00 50 00 00 54 02 00
0DD8- 55 0A 00 55 0A 00 05 0A
0DE0- 00 05 0A 00 05 0A 00 05
0DE8- 0A 00 05 0A 00 01 08 00
0DF0- 54 02 00 50 00 03 0E 00
0DF8- 00 00 00 00 00 00 20 01
0E00- 00 28 05 00 2A 15 00 22
0E08- 11 00 2A 15 00 2A 15 00
0E10- 22 11 00 22 11 00 02 10
0E18- 00 02 10 00 28 05 00 20
0E20- 01 00 0E 00 20 00 32 00
0E28- 44 00 56 00 68 00 7A 02
0E30- 08 12 01 12 01 54 00 54
0E38- 00 10 00 2A 01 02 01 00
0E40- 00 02 08 08 00 08 00 08
0E48- 00 08 00 08 00 3E 00 1C
0E50- 00 08 00 02 08 A8 85 D4
0E58- 82 A8 85 D4 82 A8 85 28
0E60- 05 50 02 28 05 02 08 A8
0E68- 85 D0 82 A0 81 40 01 40
0E70- 01 40 01 40 01 40 01 02
0E78- 08 10 02 20 01 40 00 40
0E80- 00 40 00 20 01 20 01 00
0E88- 00 02 08 40 07 60 0F 60
0E90- 0F 40 07 00 00 00 00 00
0E98- 00 00 00 02 08 80 8A C0
```

```
0EA0-  AA  C0  AA  00  8A  00  00  00
0EA8-  00  00  00  00  00  00  0E  00
0EB0-  1C  00  2A  00  38  00  46  00
0EB8-  54  00  62  02  06  10  00  10
0EC0-  00  54  00  56  01  02  01  7E
0EC8-  01  02  06  20  00  20  00  28
0ED0-  01  2C  03  04  02  7C  03  02
0ED8-  06  40  00  40  00  50  02  58
0EE0-  06  08  04  78  07  02  06  00
0EE8-  01  00  01  20  05  30  0D  10
0EF0-  08  70  0F  02  06  00  02  00
0EF8-  02  40  0A  60  1A  20  10  60
0F00-  1F  02  06  00  04  00  04  00
0F08-  15  40  35  40  20  40  3F  02
0F10-  06  00  08  00  08  00  2A  00
0F18-  6B  00  41  00  7F  00  0E  00
0F20-  18  00  22  00  2C  00  36  00
0F28-  40  00  4A  02  04  03  00  03
0F30-  00  03  00  03  00  02  04  06
0F38-  00  06  00  06  00  06  00  02
0F40-  04  0C  00  0C  00  0C  00  0C
0F48-  00  02  04  18  00  18  00  18
0F50-  00  18  00  02  04  30  00  30
0F58-  00  30  00  30  00  02  04  60
0F60-  00  60  00  60  00  60  00  02
0F68-  04  40  01  40  01  40  01  40
0F70-  01  00  0E  00  28  00  42  00
0F78-  5C  00  76  00  90  00  AA  03
0F80-  08  20  00  00  22  04  00  22
0F88-  04  00  28  01  00  28  01  00
0F90-  20  00  00  22  04  00  22  04
0F98-  00  03  08  40  00  00  44  08
0FA0-  00  44  08  00  50  02  00  50
0FA8-  02  00  40  00  00  44  08  00
0FB0-  44  08  00  03  08  00  01  00
0FB8-  08  11  00  08  11  00  20  05
0FC0-  00  20  05  00  00  01  00  08
0FC8-  11  00  08  11  00  03  08  00
0FD0-  02  00  10  22  00  10  22  00
0FD8-  40  0A  00  40  0A  00  00  02
0FE0-  00  10  22  00  10  22  00  03
0FE8-  08  00  04  00  20  44  00  20
0FF0-  44  00  00  15  00  00  15  00
0FF8-  00  04  00  20  44  00  20  44
```

```
1000- 00 03 08 00 08 00 40 08
1008- 01 40 08 01 00 2A 00 00
1010- 2A 00 00 08 00 40 08 01
1018- 40 08 01 03 08 00 10 00
1020- 00 11 02 00 11 02 00 54
1028- 00 00 54 00 00 10 00 00
1030- 11 02 00 11 02 00 85 65
1038- A5 45 85 C5 25 65 A5 E5
1040- 00 02 0C 0C 16 16 16 20
1048- 20 20 20 00 00
```

程式5-6的機械碼表（請存入畫面位址表才能執行）

本章提要

電玩遊戲很有趣，可是就好像缺少那麼一點智慧性似的，大概是沒有將電腦的特點完全的發揮出來吧！因此進一步的，我們要告訴您：

1. 最新的動畫程式——Advanture Game 。
2. 如何利用 Apple 磁片來儲存上百幕的畫面！
3. 如果將這些畫面加上第 5 章的動畫人物組成一連串的故事………（西遊記？天方夜譚？）
4. 這樣的圖形工具，再加上交談式的 BASIC 程式，便很容易造出極具吸引力的教學程式了！

6 動畫設計的策略與實現(二) ADVANTURE GAME

有人說 Arcade Game 是動手不動腦的遊戲（是嗎？），所以………

6-0 什麼是Advanture Game?

很久很久以前，有一個國王………。您是否想過，將西遊記、天方夜譚這樣的童話故事以 Apple 的畫面表現出來？配合上各種場景、文字、聲音，就是一套標準的動畫了，並且這種動畫比起電影、電視，有一個最

大的好處是其〃交談式〃的功能，所以故事演出不是螢幕上各角色的專利，使用者更是主要的參與者，因此這也是電腦輔助教學最具吸引力的地方：學習，不是單方向的灌輸，學習者（學生）才是整個學習事件的焦點，學習者的參與感，永遠是教學成敗的絕對關鍵！所以教室之所以不能像電影院，是因爲電影放映機不能知道學生的反應，但是電腦都知道（打瞌睡也包含在內）！

所以生物教師可以編一套教材：如果地球上的第一個生物不是由水中出現，結果是怎樣？而科幻小說的作家，可以經由 Apple 問遊戲者：面對著強大的黑洞吸引力，太空艦隊長應該如何處置？這便是繼電玩（Arcade Game）之後，更新穎的 Advanture Game！經由儲存在磁片中相當多數的畫面，以及一組交談程式，來造出既離奇又有趣的結果：沒有一定結局的故事！一切的故事演進，取決於使用者的交談過程，不同使用者的不同決策，可以導致完全不同的情節發展！這樣的程式，遠比電玩遊戲更具創造性、思考性與教育意義了。

當然，怎樣寫一個教學或科幻程式，不是本書所要交代的事情，我們要說的是，怎樣造出衆多豐富的畫面內容，以及最有效率的運用！

6-1 LUBAR的構想——高解像畫面佔8K？

如果一個高解像畫面佔了 8 K 位元組（33個磁區），那麼一片 5 吋磁片便只能儲存13個畫面了（計入其他主程式及系統使用區），一本 300 頁的圖畫册至少要二、三十張磁片才足以存放，如此未免太不實際了。因此動畫程式的一個主題便是如何有效的使用磁片空間，至少一張單面磁片，可以裝下 100 ～ 200個畫面才算合理吧！

畫面縮法PACKING

最早提出有關 Apple 畫面壓縮的想法者，首推Dave Lubar 1980年發表於Creative Computing 的一篇文章，而此後的許多改良版本，則均源自於 Lubar 的基本構想。什麼是畫面壓縮法呢？那就是說一幅 8 K的畫面，並非 8 K 均存有不同的資料，譬如一個星光燦爛的夜晚，天空的星星才是重要的資料，至於黑色的夜空則只不過是一些重複的 0 而已。所以對於這樣的畫面，我們只要將星星位置上的圖形資料保存，而其它黑色背景只要記得填 0 就好了，這樣 8 K 的圖形大概用不到幾十個位元組來存放吧！這便是Lubar的構想：連續出現的資料（同樣顏色的區域），只要記下是什麼顏色、多少位元組（圖形基本單位事實上是位元組而不是點）便可以了，譬如一條水平線上的圖形資料是AA AA 21 21 21 21 21 21 03 ，則以〃兩個A A 、6 個21、1 個03來記載這條水平線的資料，是比將整組資料本身列出更簡短吧！這便是圖形壓縮的原理。

彩色的問題

但是彩色又是件麻煩的事情，我們知道彩色的畫面是這樣的　（設$b_7 = 0$）：

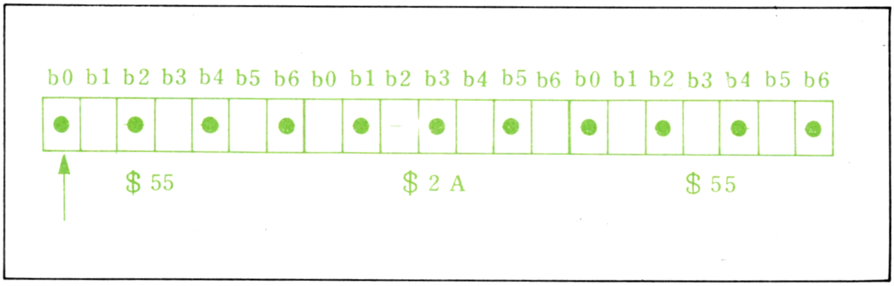


圖 6 - 0　紫色的水平線

所以一條紫色的水平線，其位元組是$55 $2A $55 $2A $55………相間變化的，如此對於一個紫色的天空，我們不但不能壓縮圖形資料，反而是(01) 55 (01) 2A (01) 55 (01) 2A………多佔了一倍空間(括號內數字表示個數)。解決的方法是將奇數位元組與偶數位元組分開，然後個別壓縮，如此便可造出如：(0A) 55 (0B) 2A ……這樣簡化的結果了。至於由磁片取回資料時，只要依奇偶順序在高解像位址上填入10個55、11個2A……便可以了。

LUBAR的程式

雖然前述的方法看似很好，可是對於像AA BB 42 53這樣的資料，却反而佔用了雙倍的記憶空間(即01 AA 01 BB 01 42 01 53)，因此對於某些較多變化的畫面，非但不能壓縮資料，反而是澎漲呢！再仔細的來考慮我們的資料形態，其實對於那種各自不同的位元組，開頭的個數位元組(即01)根本是沒有用處的，因爲反正這些位元組都只是出現一次而已，在資料取回時只要我們知道這些位元組是各自不同的，一一將之放到畫面位址上也就可以了。所以一旦有辦法知道它們是各別不同，每個資料位元組前面的01便可以去掉不用，自然就減少許多佔用空間了。但是怎樣知道是重複的位元組還是個別不同的位元組？LUBAR 的方法是這樣的：個數位元組的最高位元若爲0，則表示接下來的位元組是重複的，最高位元若爲 1，則表示按下來的位元組是個別不同的。因此：

<u>84</u> 33 2A 40 51 <u>06</u> 23 <u>82</u> 42 51 <u>02</u> 21

解碼之後是

33 2A 40 51 23 23 23 23 23 23 42 51 21 21

圖 6 - 1 圖形壓縮的編碼與解碼

圖 6 － 1 當中，84是第一個位元組，因此當然是個數位元組，其最高位元爲 1 ，所以接下來的是個別不同的資料位元組，最高位元以外的 7 個位元值爲 4 （ 84的 4 ），所以接下來的相異位元組有 4 個，解碼時便將這接來的 4 個位元組一一讀入一次（相異不重複）。在 4 個相異位元組之後接下來的是06，06的最高位元爲 0 ，因此接下來那個位元組是重複的，並且重複了 6 次，所以解碼之後有 6 個23，在23之後是82，因此42 與51只讀入一次（相異不重複），接著是02，所以21有兩個，依此類推。由此看來，位元組重複次數最多可記載 127 次，若超過便要分兩組記錄，譬如 130 個 AA ，便要記成 7F　AA　03　AA　，若記成 82 AA，則會誤當成相異位元組處理。爲了充份記載重複位元組，其實 0 ～7F是 128 個數，所以 LUBAR 將00當成 1 次重複（其實不會用到），01當成 2 次重複，所以7F便是說有 128 個重複的位元組了。有關 LUBAR 的程式，由於接著有更好的改良，所以我們列於附錄 D 以供參考。

6 2 HOLLE的方式

Lubar 的構想確實能大大的增加 Apple 磁片的畫面儲存量，平均說來一張磁片的畫面儲存是可以增加到30～40個畫面，一個原來佔用34磁區的畫面平均是佔7 ～25個磁區而已。不過，我們仍可進一步改良！

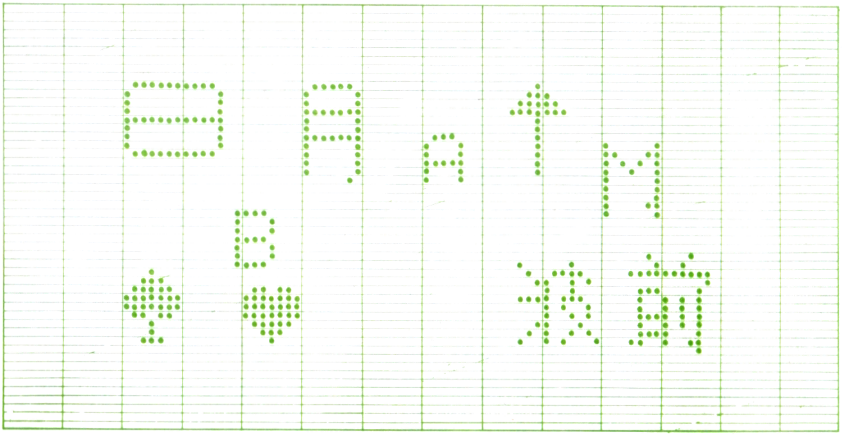


圖 6 - 2　一個高解像畫面圖形的一部份

由圖 6 — 2 我們可以發現，圖形資料重複區是在畫面上整塊出現的，也就是說，這些重複位元組是依畫面位置座標而不是依記憶位址順序排列的，所以在做壓縮及擴展（取回圖形）時，如果依記憶位址取資料，則由於 Apple 畫面垂直方向的位址不連續性，會使得重複位元組不易取到，譬如 6 — 2 的上方共有 9 條水平線是連續的空白圖形（因此有 9 ×40＝360 個重複位元組），但若以位址順序來看的話 *，則這些連續的位元組便每40個一組（一條水平線）的被拆散了。

* 參考附錄 A

```
!L
*
************************
* PACK-UNPACK ROUTINE *
************************
*
LINES       EQU  192             ;水平線數
TBL         EPZ  $00
```

```
XCOORD     EPZ $02
YCOORD     EPZ $03
OFFSET     EPZ $04
GBAS       EPZ $05
PREV       EPZ $07          ；前一位元組資料
REPEAT     EPZ $08          ；重複次數
***********************
* START UNPACKING !!! *
***********************
*
           ORG $A00
UNPACK     LDA $E6
           ORA #$04
           STA GBAS+1       ；$2400（X=0，Y=1）
           LDX #$01
           STX OFFSET
           LDY #$00
           STY XCOORD
           STY GBAS
           STY REPEAT
PROC       LDA (TBL),Y      ；取資料
           BNE STORE        ；互異位元組
           INC TBL
           BNE GETREP
           INC TBL+1
GETREP     LDA (TBL),Y      ；取計數
           STA REPEAT
           INC TBL
           BNE GETPREV
           INC TBL+1
GETPREV    LDA (TBL),Y      ；取資料
           STA PREV
DOREP      LDA PREV
           DEC REPEAT
STORE      LDY XCOORD
           STA (GBAS),Y
           INX
           INX
           CPX #LINES       ；底線？
           BCC YOK
           INC XCOORD
           LDY XCOORD
           CPY #40          ；一回完畢？
           BCC XOK
           DEC OFFSET       ；第二回也完畢？
```

```
               BMI  DONE
               LDY  #0
               STY  XCOORD
XOK            LDX  OFFSET
YOK            JSR  BASEC
               LDY  REPEAT   ；重複？
               BNE  DOREP    ；是，則到DOREP
               INC  TBL
               BNE  PROC
               INC  TBL+1
               BNE  PROC
DONE           RTS
BASEC          TXA           ；水平線起始位址計算
               AND  #$C0
               STA  GBAS
               LSR
               LSR
               ORA  GBAS
               STA  GBAS
               TXA
               STA  GBAS+1
               ASL
               ASL
               ASL
               ROL  GBAS+1
               ASL
               ROL  GBAS+1
               ASL
               ROR  GBAS
               LDA  GBAS+1
               AND  #$1F
               ORA  $E6
               STA  GBAS+1
               RTS
*
*********************
* START PACKING !!! *
*********************
*
PACK           LDY  #1
               STY  OFFSET
               STY  YCOORD
               DEY
               STY  XCOORD
               LDA  $E6
```

```
                ORA #$04
                STA GBAS+1
                STY GBAS
                LDA (GBAS),Y
                BNE MORE
                ORA #$80
MORE            LDX #$01
                STX REPEAT        ; 計數＝1
                STA PREV
TRAVEL          LDY XCOORD
                LDX YCOORD
                INX
                INX
                CPX #LINES        ; 底線？
                BCC YOL           ; 否，至 YOL
                INY               ; 向右延伸
                CPY #40           ; 一線完畢？
                BCC XOL           ; 否，至 XOL
                DEC OFFSET        ; 兩回 PACK 完畢？
                BMI NOTEQ         ; 是，至 NOTEQ
                LDY #0
XOL             STY XCOORD
                LDX OFFSET
YOL             STX YCOORD
                JSR BASEC
                LDA (GBAS),Y      ; 由畫面位址取資料
                BNE PROC2
                ORA #$80          ; 將 $00 換成 $80
PROC2           CMP PREV          ; 相同的資料？
                BNE NOTEQ         ; 不同，至 NOTEQ
                INC REPEAT        ; 重複次數加一
                BNE TRAVEL        ; 最多可計至 256
                DEC REPEAT
NOTEQ           PHA               ; 存新編碼資料
                LDY #0
                LDX REPEAT
                BEQ BIG
                CPX #4            ; 是否重複 4 次以上
                BGE BIG           ; 若是，則編碼，小於 4 次
                LDA PREV          ; 則不編碼，否則不划算
LITTLE          JSR OUT
                DEX
                BNE LITTLE
                BEQ FRESH
BIG             TYA               ; 以 00 做重複指示
```

```
             JSR  OUT         ; 存 00          }
             TXA              ; 存重複字數     }  00  PP  DD
             JSR  OUT                          }
             LDA  PREV        ; 存資料         }
             JSR  OUT
FRESH        PLA              ; 取新資料
             BIT  OFFSET      ; 結束？
             BPL  MORE        ; 否，至more
             RTS
OUT          STA  (TBL),Y     ; 將結果造出至指定位址
             INC  TBL
             BNE  DONE2
             INC  TBL+1
DONE2        RTS
             END
```

程式 6 - 0　圖形壓縮與還原程式

另外，觀察圖 6 — 2，我們可以發現垂直方向的重複機會比水平大，譬如〃日、月、M、↑〃等字形垂直看來皆有許多位元組是重複相同的，但水平方向却無。不過彩色仍然是問題，記得第一章的彩色製造法嗎？連續使用橙、綠相間的水平線便能造出黃色的效果來，但是這樣的黃色圖形却造成了垂直方向的不重複！幸好，我們仍舊可以使用類似 LUBAR 的奇偶壓縮法，先對奇數位元組取資料，再對偶數位元組取資料，便可以解決上述的難題。

HOLLE 還將 LUBAR 的編碼方式稍加修改，對於不重複的位元組，乾脆省掉最前面的個數位元組，而改以直接列出，至於重複位元組則以00做起始指示位元組，接著才是個數位元組，最後是被重複的那個資料位元組。雖然相異位元組少了個數指示位元組，而重複位元組則多了00做指示位元組，如此看來似乎沒有什麼差異，其實這樣做有兩個好處，第一是重複位元組的記載可以由 128 個增加到 256 個，第二是用 LUBAR 的方法，當重複位元組很少而相異者甚多的時候，每 128 個相異位元組之前便要多一個個數位元組，如此可能造出比原資料還浪費空間的情形，使用HOLLE

的改良方式便不會有上述的情形了。用這種方法時，當位元組的重複次數不超過 3 次時，則將之編碼便不划算了，因爲譬如 2 次重複的 A A 位元組，當成互異位元組看待時，僅用到 2 個位元組，而當成重複位元組看待時反而變成 00　01　AA 共 3 個位元組了。程式 6 － 0，基本上便是將HOLLE 的方式稍加修飾而成的，使用這個程式一個原本佔用34磁區的 8 K 畫面，壓縮約只有 2 ～ 3 個磁區，最多很少超過 5 個磁區的，一張磁片共可存放多少畫面呢？

6-3 畫面壓縮　還原使用例

使用程式 6 － 0時，必須設定 3 個位址，其中兩個是$00與$01，這兩個位址用來指出壓縮後圖形資料的起始位址，當使用圖形還原程式時，這兩個位址用以指出待還原圖形資料現在的起始位址，其位址存放方式是低位元組在先（存於 $00），高位元組在後（存於 $01）。另外一個必須設定的位址是$E6，若此位址的內含值是 $20（32），則所處理的是高解像第一頁區的圖形資料，如果內含值是 $40（64），則所將壓縮或還原的是高解像第二頁區的圖形資料。 $E6可以被用來做頁區指定，因此您可以在顯示第一頁區時，還原第二頁區的畫面，以造成快速畫面變化的效果。要注意的是，一旦壓縮或還原程式執行之後，（$00，$01）所含的不是畫面資料存放的起始位址，而是終止位址了。當然壓縮及還原是要分別由程式 6 － 0 的 PACK 及 UNPACK進入點呼叫使用的。

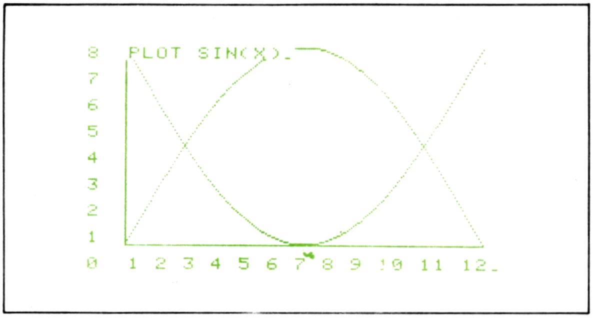
使用PACK-UNPACK的程式

底下的程式，可以很方便的來使用 PACK － UNPACK這兩個機械語言副程式，您馬上就可以用程式 1 － 2 所造出的圖形來試試看，至於要造出怎樣的圖形，怎樣的故事，相信您也同意這不是我該說的了。

```
]LIST

0  REM  ********************************
1  REM  ***  PACKER-UNPACKER HANDLER  ***
2  REM  ********************************
10 D$ =  CHR$ (4):PACK = 4220:UNPACK = 409
     6:LOC1 = 24576
20  PRINT D$;"BLOAD PACK-UNPACK.OBJ"
30  HOME : HTAB 10: INVERSE : PRINT "   PA
     CKER - UNPACKER   ": PRINT : PRINT
40  NORMAL : HTAB 10: PRINT "PACK OR UNPAC
     K ? (P/U) "
50  GET P$: IF P$ <  > "P" AND P$ <  > "U"
      THEN 50
60  PRINT P$: PRINT : PRINT
70  PRINT "HI-RES SCREEN (1/2)?": GET S$
80  IF S$ <  > "1" AND S$ <  > "2" THEN 70

90  PRINT S$
100  INPUT "FILE NAME: ";F$: IF F$ = "" THEN
      100
110  PRINT : PRINT : PRINT "OK (Y/N)?"
120  GET A$: IF A$ <  > "Y" AND A$ <  > "N
     " THEN 120
130  IF A$ = "N" THEN 30
134  REM  ***********************
135  REM  *** SET SCREEN MODE ***
136  REM  ***********************
140  POKE  - 16304,0: POKE  - 16297,0: POKE
      - 16302,0: POKE 230,32
150 : IF S$ = "2" THEN  POKE  - 16299,0: POKE
     230,64
160  IF P$ = "P" THEN 200
170  GOTO 300
```

```
200  REM  ***  PACK & SAVE  ***
210  POKE 0,LOC1 - ( INT (LOC1 / 256) * 25
     6): POKE 1, INT (LOC1 / 256)
220  CALL PACK
230 LN =  PEEK (0) +  PEEK (1) * 256 - LOC
     1: REM    LENGTH
240  PRINT : PRINT D$;"BSAVE";F$;".PACK,A"
     ;LOC1;",L";LN
250  GOTO 400
299  REM  ***********************
300  REM  ***  UNPACK & LOAD  ***
301  REM  ***********************
320  PRINT : PRINT D$;"BLOAD";F$;".PACK"
325  POKE 0,LOC1 - ( INT (LOC1 / 256) * 25
     6): POKE 1, INT (LOC1 / 256)
330  CALL 4096
340  GOTO 430
399  REM  ***********************
400  REM  *** ANYTHING ELSE ? ***
401  REM  ***********************
410  POKE  - 16301,0
420  VTAB 21: PRINT "HIT ANY KEY TO CONTIN
     UE ": GET A$
430  TEXT : HOME : VTAB 10: INVERSE
440  PRINT "   QUIT OR CONTINUE (Q/C)?   "
     ;
450  GET C$: NORMAL
460  IF C$ <  > "Q" AND C$ <  > "C" THEN 450

470  IF C$ = "C" THEN 30
480  HOME
481  REM  ***********************
482  END
483  REM  ***********************
```

A APPLE II 高解像畫面結構



A 0 顏色的規則

APPLE Ⅱ 參考手册，對於高解像畫面上，圖點的彩色規則的說明，相當的淸楚明白：

1. Apple 以＄2000 至＄3FFF 這段記憶位址區間，來存放高解像圖形資料，稱爲高解像螢幕位址區。只要將螢幕設定成高解像畫面，則所有高解像螢幕位址區的內容便直接表現到螢幕上來。
2. 高解像畫面上的每個圖點，均對應到螢幕記憶位址內的某一特定的位元（bit），如果將“1”存入到這個位元，則該位元所對應到

的圖點便“點亮”了，如果將“0”存入則該圖點便被熄滅（黑色）。

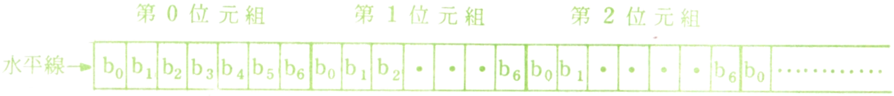
3.螢幕位址區的每一位元組　所含8個位元的最高位元（b_7）並不用來對應於任何圖點，而是用來控制彩色相位。

4.一個圖點被點亮之後，其顏色是由該點在畫面上的位置及所屬位址的最高位元依圖A－0來決定的。

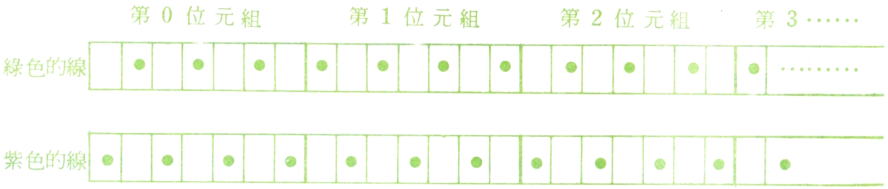
最高位元 \ 位置	偶數點	奇數點	連續兩點
0	紫	綠	白
1	藍	橙	白

圖A-0　畫面位置與顏色的關係

由前述的規則，我們可以歸納出底下的位置關係來：



圖A-1　畫面上水平方向圖點與位址的結構



圖A-2　彩色水平線的結構

由圖A－2，我們可以發現，雖然是看起來一整條的綠色水平線，其位址內含值却是由$2A、$55、$2A………這樣相間造成的(而不是直覺上想像的由同一種數值構成的)，這便是Apple特出的畫面構造(也是令設計者傷腦筋的地方)，如果要將綠色($2A)的圖形移向右邊一個位元組，是不可以直接將$2A值存入下一位址的，必須是存入$55才行。圖A－3便是一個彩色的圖形，其中不參與畫圖的b_7(選色用，後述)設為1，請仿此練習其它圖形，以熟習其畫法，並且也請試著移位看看！

●																81	80	80
●		●														85	80	80
●		●														85	80	80
●		●		●		●		●								D5	82	80
●		●		●						●		●		●		95	A8	81
●		●		●		●		●								D5	82	80
b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_0		位	元	組

圖A－3　彩色的飛機

A 1 高解像記憶位址與座標位置的關係

我們提到高解像的每一圖點，對應到記憶位址上的一個位元，可是到**底是怎樣**的一個對應法呢？我們發現到這個對應的法則並不太簡單……

圖A－4 高解像畫面的記憶位址

0	8192- 8231	48	8960- 8999	96	8744- 8783	144	8528- 8567
1	9216- 9255	49	9984-10023	97	9768- 9807	145	9552- 9591
2	10240-10279	50	11008-11047	98	10792-10831	146	10576-10615
3	11264-11303	51	12032-12071	99	11816-11855	147	11600-11639
4	12288-12327	52	13056-13095	100	12840-12879	148	12624-12663
5	13312-13351	53	14080-14119	101	13864-13903	149	13648-13687
6	14336-14375	54	15104-15143	102	14888-14927	150	14672-14711
7	15360-15399	55	16128-16167	103	15912-15951	151	15696-15735
8	8320- 8359	56	9088- 9127	104	8872- 8911	152	8656- 8695
9	9344- 9383	57	10112-10151	105	9896- 9935	153	9680- 9719
10	10368-10407	58	11136-11175	106	10920-10959	154	10704-10743
11	11392-11431	59	12160-12199	107	11944-11983	155	11728-11767
12	12416-12455	60	13184-13223	108	12968-13007	156	12752-12791
13	13440-13479	61	14208-14247	109	13992-14031	157	13776-13815
14	14464-14503	62	15232-15271	110	15016-15055	158	14800-14839
15	15488-15527	63	16256-16295	111	16040-16079	159	15824-15863
16	8448- 8487	64	8232- 8271	112	9000- 9039	160	8784- 8823
17	9472- 9511	65	9256- 9295	113	10024-10063	161	9808- 9847
18	10496-10535	66	10280-10319	114	11048-11087	162	10832-10871
19	11520-11559	67	11304-11343	115	12072-12111	163	11856-11895
20	12544-12583	68	12328-12367	116	13096-13135	164	12880-12919
21	13568-13607	69	13352-13391	117	14120-14159	165	13904-13943
22	14592-14631	70	14376-14415	118	15144-15183	166	14928-14967
23	15616-15655	71	15400-15439	119	16168-16207	167	15952-15991
24	8576- 8615	72	8360- 8399	120	9128- 9167	168	8912- 8951
25	9600- 9639	73	9384- 9423	121	10152-10191	169	9936- 9975
26	10624-10663	74	10408-10447	122	11176-11215	170	10960-10999
27	11648-11687	75	11432-11471	123	12200-12239	171	11984-12023
28	12672-12711	76	12456-12495	124	13224-13263	172	13008-13047
29	13696-13735	77	13480-13519	125	14248-14287	173	14032-14071
30	14720-14759	78	14504-14543	126	15272-15311	174	15056-15095
31	15744-15783	79	15528-15567	127	16296-16335	175	16080-16119
32	8704- 8743	80	8488- 8527	128	8272- 8311	176	9040- 9079
33	9728- 9767	81	9512- 9551	129	9296- 9335	177	10064-10103
34	10752-10791	82	10536-10575	130	10320-10359	178	11088-11127
35	11776-11815	83	11560-11599	131	11344-11383	179	12112-12151
36	12800-12839	84	12584-12623	132	12368-12407	180	13136-13175
37	13824-13863	85	13608-13647	133	13392-13431	181	14160-14199
38	14848-14887	86	14632-14671	134	14416-14455	182	15184-15223
39	15872-15911	87	15656-15695	135	15440-15479	183	16208-16247
40	88[illegible]- 8871	88	8616- 8655	136	8400- 8439	184	9168- 9207
41	9856- 9895	89	9640- 9679	137	9424- 9463	185	10192-10231
42	10880-10919	90	10664-10703	138	10448-10487	186	11216-11255
43	11904-11943	91	11688-11727	139	11472-11511	187	12240-12279
44	12928-12967	92	12712-12751	140	12496-12535	188	13264-13303
45	13952-13991	93	13736-13775	141	13520-13559	189	14288-14327
46	14976-15015	94	14760-14799	142	14544-14583	190	15312-15351
47	16000-16039	95	15784-15823	143	15568-15607	191	16336-16375

高解像位址＝8192＋〔INT(Y／8)MOD8)〕＊128＋INT(Y／64)＊40＋(Y MOD 8)＊1024＋INT(X／7)
其中MOD的計算法是：A MOD B＝A－INT(A／B)＊B
X＝0－279，Y＝0－191

圖A-5 由畫面座標(X，Y)算出對應的記憶位址

圖A－4就是高解像畫面每一水平線上的記憶位址，而圖A－5的便是用來計算圖A－4表格的公式，你可以代入一些數值，譬如X＝12，Y＝13算出來是第13條水平線上的第12個圖點所屬的位址，其結果是13441。當您更仔細的核對圖A－5及A－4時，會發現到某些介於＄2000～＄3FFF（或＄4000～＄5FFF）間的位址並不存在於圖A－4內，這是Apple的特殊畫面結構，使得自128號～191號水平線每條水平線之後還隱藏了8個位址，並沒有使用，因此共有8×(191－127)＝512位元組未使用，這些位址由於是8個一組，而每組之間的位址並不連續，因此不適於用來寫程式，但是可以做爲變數或陣列位址使用區。用另外一個角度來看，高解像畫面每條水平線是由40位元組構成，因此共有40位元組×192線＝7680位元組，而8K的畫面是8192位元組，恰好是512位元組未使用到。

A 2 Apple Ⅱ的畫面解析度是560點？(神秘的b_7位元)

最初的Apple Ⅱ ver.0高解像畫面是只有兩色的(即紫與綠)，而後來才利用b_7來造出另外橙、紫兩個顏色，有關其詳細動作及演進請參考"APPLE Ⅱ硬體線路分析——附故障檢修"(波前出版)一書，當有詳盡的解說，此處我們僅作提要說明。

Apple的紫綠顏色在畫面上的水平分佈是這樣的：

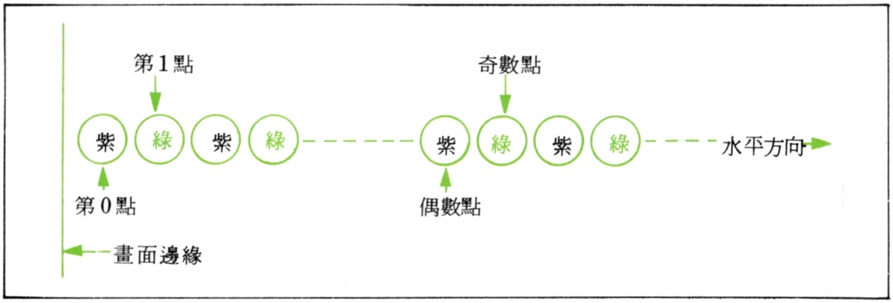
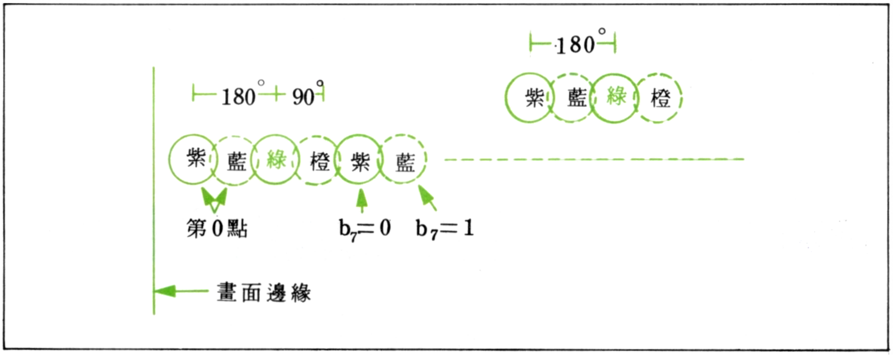


圖 A - 6　畫面點的顏色分佈圖（ $b_7 = 0$ ）

只要在偶數點的位置上畫點，便造出一個紫色點來，而在奇數位置上畫點，則爲綠色，畫面之所以紫綠相間，是因爲二者相位差別180°（互補色）所致。

由 A — 0 節所述的顏色規則知道，當每一位元組的第 7 位元變成 1 時，則偶數位置的點變成藍色，而奇數位置的點變成橙色，而這種變化是由於第 7 位元所造成的90°相位移而致之，所以顏色分佈看起來是：



圖A - 7　畫面點的顏色分佈圖（ $b_7 = 1$ 及 $b_7 = 0$ ）

圖A — 7 是很容易瞭解的：既然圖A — 6 的藍色與紫色是相位差180°，那麼由b_7來造成 90°相位移後的橙色及綠色點的位置便是交互間叉在藍色與

紫色點的位置之間了。

圖A－6的畫點位置，由Apple 手册上的說明，知道是280個點，這樣說來圖A－7的所有畫點位置便是280×2＝560位置，所以事實上：

1. Apple 高解像畫面的解析度是560點
2. 要達到560點的解像度，必須用b_7（每一位元組的最高位元）來移位。
3. Apple 高解像畫面最多可畫280點。
4. Apple 高解像畫面最多可畫140個彩色點。
5. Apple 高解像畫面一次必須存一位元組而不是位元。
6. Apple 高解像畫面最多只可使用水平40個位元組。

這便是我們的結論了，雖然是黑白圖形，使用b_7却可以使圖點移動1／560的畫面寬度。或許b_7對您來說無甚用處，其實許多職業設計的圖形或字形，便多是用b_7來做最細步（1／560畫面位移）的邊緣調整，以達到令人讚嘆的效果！

B 偵錯程式

組合語言的程式，往往很難於偵錯，因此，我們介紹您一些偵錯及避免錯誤的方法，但是，無論如何，良好的程式設計習慣，永遠是寫好程式的不二法門。

B-0 程式設計的原則

1. 要先完全瞭解設計的目標，尤其是龐大的程式，絕不可冒然從事。
2. 瞭解之後，便可開始思考對策。所有的問題及解決方法，可以用文句敍述及流程方塊來輔助思考。
3. 先擬定大原則，再解決細節問題。
4. 程式寫好後，最好先自行模擬執行一遍，將每一關鍵點的預期結果記錄下來，最才可上機。

經由上列的 4 個步驟，您已對整個工作有了相當的掌握了。

B-1 偵錯程式

這是Apple Graphic Arcade Game Design 一書(R3)所載的一個相當簡短的偵錯程式，對於臨時附加到程式裡使用，可說是相當的方便。

```
*****************
* DEBUG ROUTINE *
*****************
*
*          .........
*          .........
*
           LDA KEY          ;KEY PRESSED?
           BPL OUT          ;IF NOT , GET OUT
           CMP #$9B         ;ESC KEY?
           BNE OUT          ;NO, GET OUT
STEP       BIT RESETKEY     ;RESET KEY
READ       LDA KEY          ;KEY PRESSED?
           BPL READ
           CMP #$A0         ;SPACE KEY?
           BNE OUT1
OUT        BIT RESETKEY
OUT1       NOP
*          .........
*          .........
```

程式B - 0　偵錯程式

此程式必須安插於您所設計的程式裡頭，如果您是將它安排在主廻圈內，則主廻圈執行一回才通過偵錯程式一次。如果要對主廻圈做更進一步的控制，則程式B－0可以做成副程式，然後以JSR DEBUG 插入到主廻圈的各構成方塊前後，這樣整的圈廻的動作就更容易掌握了。

程式的使用法是以上述方式將程式B－0安插到主迴圈後，開始執行主程式，一旦您對畫面某些動作發生疑問時，可以按下 ESC 鍵來啓動偵錯程式，偵錯程式啓動後，便將主程式投入一個等候按鍵的迴圈裡，因此畫面的動作便停止了。如果您要令程式進行下一步工作的話，這時可以按下 空格鍵 以外的任何鍵，主程式便可以再進行其工作，直到主迴圈又繞迴到偵錯程式爲止。所以，基本上這是一個單步迴圈（或單步方塊）的偵錯程式，而不是眞正的單步偵錯程式。雖然啓動了偵錯程式後，每次按鍵（ 空格鍵 除外）畫面看似只有很小的變化（移動），但事實上 Apple 已執行了很複雜的機械語言指令了。

說起來，這種方法的優點是容易對畫面動作做較明顯的觀察（因爲每一偵錯步驟間，已有許多指令被執行，因而畫面變化也較明顯），但是其優點也正是其缺點：不方便更細步的來觀察程式執行過程。

B-2 LISA的TRACE程式

在LISA磁片的第二面，有一個叫TRACE 的程式，這個程式的功能類似於CP／M的DDT或 CROMENCO 的 DEBUG程式。這是一個能夠追踪程式每一步驟的偵錯程式（有關其用法請參考LISA 使用手册，此處不予詳述），這一類型的偵錯程式的缺點是因爲它以文字幕來顯示其執行過程（將每一步驟的A、X、Y、P、S及旗號變化等均顯示出來），一旦動畫程式以高解像畫面取代了文字幕之後，則其顯示的訊息便無從觀察了。也就是說，當您使用LISA 的TRACE 來偵錯時，雖然可以看到每一動作細節，但是對於使用高（低）解像畫面的程式，却不甚方便。雖然我們可以用印字機代替文字幕來做訊息顯示的工具，但是在某些迴圈的場合（譬如等候按鍵、延時等），印字機會印完整個迴圈的動作細節，而這時對我們來說是一段很長而無用的資料。

經驗上說來，一個程式設計者眞正借助於偵錯程式的場合並不多，細

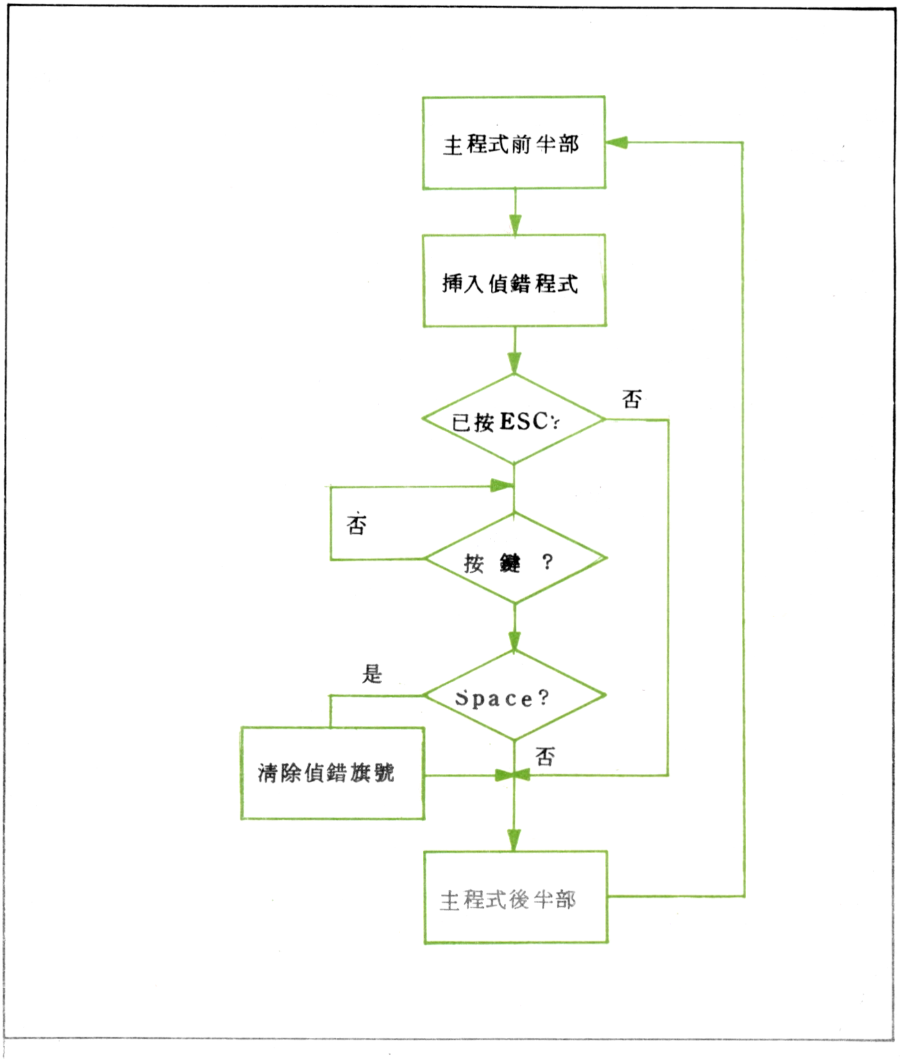
心的做好設計前的準備工作，防止錯誤於未然，才是優秀的程式設計者吧！雖然默菲定律告訴我們，任何的複雜的程式設計，永遠有發生錯誤的可能，但是對於整體設計有全盤瞭解的您，也永遠可以找出問題的根源，而成功的寫好程式吧！

B-3 改良的偵錯程式

程式B-0本身的設計有些缺失，如果您眞正插入到動畫程式中，則由於動畫走完一次總迴圈須要相當長的時間，因此掃描時間長，可能當我們按鍵時，不是輪到執行程式B－0的部份，譬如，正在繪圖，如果此時按下 ESC 鍵便沒有偵錯效果了。改良後的程式是B－1，其最大改變是當按鍵時，雖然主迴圈正在執行程式其它部份，而不檢知到按鍵，但是當主迴圈後來轉到程式B－1部份時，第9行～第10行仍然可以檢知到是否按 ESC 鍵。

```
 2 ****************************
 3 * DEBUGER ROUTINE BY S.U.M *
 4 *   - ALL RIGHT RESERVED -   *
 5 ****************************
 6 *
 7 DEBUGER    LDA FLAGO
 8            BNE KO
 9            LDA KEY
10            CMP #$1B
11            BNE OUT
12            LDA #$01
13            STA FLAGO
14 KO         LDA KEY
15            BPL KO
16            CMP #$A0
17            BNE OUT
18            LDA #$00
19            STA FLAGO
20 OUT        BIT RESETKEY
```

程式B-1　改良的偵錯程式



圖B-0　偵錯程式使用法

C 電玩的聲音副程式

> 小精靈如果沒有聲音結果怎樣呢？一個不發出聲響的長生鳥，其遊戲吸引力馬上就失色三分，可是，電玩的音響效果是怎樣製造的呢？

C-0 視覺與音響共存的秘密

如果你是第一次設計遊戲程式，對於在動態畫面上附加音響效果的處理上，可能會遇到一個問題：那就是當喇叭發聲時，畫面會停止動作，譬如一架急駛中的賽車，伴隨著引擎的聲音，可能就會因爲聲音副程式的加入而令畫面動作不自然了。有一個叫飛機大戰（BLITZKRIEG或BLAST）的Game，是較早期（1980年版的設計），當地上的炮彈發射喇叭發聲時，空中的飛機便停頓一下再走（這個現象可以在輸入程式後，按'M'及'9'令飛機速度加快，缺點更明顯）。

同樣的，在程式5－6我們也用$FBDD一面來發聲一方面做延時，所以當爆炸發聲時，其它外星人也是停止不動的。只不過這並不造成任何問題，因為當物體爆炸時，遊戲者的注意力是在爆炸圖形，而不是畫面其他部份。可是執行程式5－6，按鈕發射子彈時，也會有另外一種類如機關槍連發的聲音，這種聲音却不造成任何畫面停頓的缺點，這種效果是怎樣造成的呢？

原來，就像畫面處理一般，Apple 的CUP對任何事情，一次只能處理一件，也就是說移動太空船的同時，就不能移動外星人，因此當CUP令喇叭發聲時，就不能做畫面處理，所以畫面動作便凍結了。不過，幸好是機械語言的速度夠快，我們可以像畫面處理一樣的，將一連串的事件快速執行，整個結果是：喇叭既可發聲，畫面動作又不受影響。

```
                 :
SOUND0     LDY     #$01
           STA     $C030
           DEY
           BNE     SOUND0
                 :
```

程式C－0　串在主程式當中的發聲程式例

在程式5－6當中，我們是LDY #$01，可能你會說這樣只不過觸動$C030（喇叭位址）一次而已，直接用STA $C030不就可以了嗎？是的，我們採用程式C－0的原因是您可以改變LDY的數值，然後聽其效果，請試著將#$01改成#$02，結果？改成#$03，結果？（為何？）

C-1 改進音響的效果

程式C－0看起來平淡無奇，所以在程式5－6當中用起來也就不甚特出，不過由於其結果簡單，而我們也是另有用意採用它的。

請在程式5－6當中，將發聲程式的LDY #$01改成LDY #$FF，結果畫面動作看來似乎也不受什麼影響，但是，當Y－暫存器的值爲#$FF 時，程式C－0的迴圈共要跑255次，而這樣的時間延時竟然不減慢畫面動作，可見機械語言的快速了，也可見我們可以寫一個更具花樣的（當然更長）聲音程式來取代程式C－0，而不會影響畫面動作的品質了。譬如，在程式5－6當中，只要太空船子彈存在畫面上，喇叭就被觸動發聲，其實你可以在程式中加上一個計數器，使子彈發射的最初幾次移動發出聲響，而在一段距離後就停止發聲，這樣更接近實際情況吧！至於其它各種音效的製造，則請參考有關文獻＊，當然最後一定要經由實際試驗才能符合您的需要。

＊ APPLE II 繪圖遊戲系統設計　第一部份　林和賢著　波前出版　APPLE II 精彩遊戲系列(2)

D LUBAR的 PACK-UNPACK程式

```
!L
;
*******************************
*  PACKER 3.0 BY DAVID LUBAR  *
*     - ONLY FOR REFERENCE -   *
*******************************
*
                ORG $1000
SCLO            EPZ 0
SCHI            EPZ 1
TABLO           EPZ 2
TABHI           EPZ 3
OFF             EPZ 4
COUNT           EPZ 5
FLAG            EPZ 6
FLAG1           EPZ 7
FIX             LDA #$0
                LDX #$08
FIX1            STA $3FF8,X
                DEX
                BPL FIX1
SET             LDA #$0
                STA FLAG
                LDA #$1
                STA OFF
SET1            LDA #$1
                STA TABLO
                LDA #$40
                STA TABHI
                LDA #0
                STA COUNT
SET2            LDA #$20
                STA SCHI
                LDA COUNT
                STA SCLO
                LDA #$0
```

```
               TAX
               TAY
      START    LDA (SCLO,X)
      LOOP     PHA
               LDA SCLO
               CLC
               ADC OFF
               STA SCLO
               BCC CONT
               INC SCHI
               LDA SCHI
               CMP #$40
               BEQ DONE
      CONT     PLA
               INY
               BMI PAGE
               CMP (SCLO,X)
               BEQ LOOP
      PAGE     PHA
               DEY
               TYA
               STA (TABLO,X)
               INC TABLO
               BNE CONT1
               INC TABHI
      CONT1    PLA
               STA (TABLO,X)
               INC TABLO
               BNE CONT2
               INC TABHI
      CONT2    LDY #$0
               BEQ START
      DONE     TYA
               STA (TABLO,X)
               INC TABLO
               BNE CONT5
               INC TABHI
      CONT5    PLA
               STA (TABLO,X)
               INC TABLO
               BNE CONT6
               INC TABHI
      CONT6    INC COUNT
               LDA COUNT
               CMP OFF
```

```
          BNE  SET2
          LDA  FLAG
          BNE  OUT
NEXT      LDA  TABHI
          LDX  OFF
          STA  FLAG,X
          LDA  OFF
          ASL
          STA  OFF
          CMP  #$8
WAY       BNE  SET1
TEST      LDY  #$1
          LDA  $7
          CMP  $8
          BCC  TEST1
          LDY  #$2
          LDA  $8
TEST1     CMP  $A
          BCC  TEST2
          LDY  #$4
TEST2     STY  OFF
          STY  $4000
          LDA  #1
          STA  FLAG
          BNE  WAY
OUT       LDA  #$FF
          STA  (TABLO,X)
*
REPACK    LDA  #1
          STA  TABLO
          STA  SCLO
          LDA  #$40
          STA  TABHI
          STA  SCHI
          LDX  #0
LOOP3     LDA  (TABLO,X)
          BEQ  SINGLE
          BMI  DONE2
          STA  (SCLO,X)
          INC  TABLO
          BNE  CNT1
          INC  TABHI
CNT1      INC  SCLO
          BNE  CNT2
          INC  SCHI
```

```
CNT2     LDA (TABLO,X)
         STA (SCLO,X)
         INC TABLO
         BNE CNT3
         INC TABHI
CNT3     INC SCLO
         BNE LOOP3
         INC SCHI
         BNE LOOP3
DONE2    RTS
SINGLE   TAY
         STA FLAG
         LDA #3
         STA FLAG1
LOOP4    INC TABLO
         BNE CNT4
         INC TABHI
CNT4     LDA (TABLO,X)
         STA (FLAG),Y
         INC TABLO
         BNE CNT5
         INC TABHI
CNT5     INY
         BMI PAGE2
         LDA (TABLO,X)
         BEQ LOOP4
PAGE2    DEY
         TAY
         ORA #$80
         STA (SCLO,X)
         INC SCLO
         BNE CNT6
         INC SCHI
CNT6     INY
LOOP5    LDA (FLAG,X)
         STA (SCLO,X)
         INC FLAG
         INC SCLO
         BNE CNT7
         INC SCHI
CNT7     DEY
         BNE LOOP5
         BEQ LOOP3
         END
```

程式D－0　LUBAR的畫面壓縮程式

```
!L
;
********************************
*  UNPACKER 3.0 BY DAVID LUBAR  *
*      - ONLY FOR REFERENCE -   *
********************************
*
              ORG $900
SCLO          EPZ 0
SCHI          EPZ 1
TABLO         EPZ 2
TABHI         EPZ 3
OFF           EPZ 4
COUNT         EPZ 5
SETUP         LDA $4000
              STA OFF
              LDA #1
              STA TABLO
              LDA #$40
              STA TABHI
              LDA #0
              TAX
              STA COUNT
SET2          LDA #$20
              STA SCHI
              LDA COUNT
              STA SCLO
START1        LDA (TABLO,X)
              BMI SING
              INC TABLO
              BNE CON1
              INC TABHI
CON1          TAY
              INY
              LDA (TABLO,X)
LOOP1         STA (SCLO,X)
              PHA
              LDA SCLO
              CLC
              ADC OFF
              STA SCLO
              BCC CON3
              INC SCHI
              LDA SCHI
              CMP #$40
```

```
         BEQ  OUT1
CON3     PLA
         DEY
         BNE  LOOP1
         INC  TABLO
         BNE  START1
         INC  TABHI
         BNE  START1
OUT1     PLA
OUT2     INC  TABLO
         BNE  CON9
         INC  TABHI
CON9     INC  COUNT
         LDA  COUNT
         CMP  OFF
         BNE  SET2
         RTS
SING     EOR  #$80
         TAY
         INY
         INC  TABLO
         BNE  LOOP5
         INC  TABHI
LOOP5    LDA  (TABLO,X)
         STA  (SCLO,X)
         INC  TABLO
         BNE  CON6
         INC  TABHI
CON6     LDA  SCLO
         CLC
         ADC  OFF
         STA  SCLO
         BCC  CON7
         INC  SCHI
         LDA  SCHI
         CMP  #$40
         BEQ  CON9
CON7     DEY
         BNE  LOOP5
         BEQ  START1
         END
```

程式D - 1　LUBAR的畫面還原程式

E 有趣的英文字母

```
*6300.646B

6300-  00 0E 00 40 00 72 00 A4
6308-  00 D6 01 08 01 3A 03 10
6310-  00 00 00 00 00 00 40 03
6318-  00 60 07 00 30 0C 00 18
6320-  18 00 1C 30 00 0C 30 00
6328-  0C 30 00 7C 3F 00 7C 3F
6330-  00 0C 30 00 0C 30 00 0C
6338-  30 00 00 00 00 00 00 00
6340-  03 10 00 00 00 00 00 00
6348-  00 07 00 40 0F 00 60 18
6350-  00 60 18 00 60 18 00 60
6358-  18 00 60 18 00 60 1F 00
6360-  60 1F 00 60 1D 00 70 3D
6368-  00 78 7F 00 00 00 00 00
6370-  00 00 03 10 00 00 00 00
6378-  00 00 00 0E 00 00 1F 00
6380-  40 31 00 60 60 00 60 60
6388-  00 60 60 00 60 60 00 60
6390-  7F 00 60 7F 00 60 60 00
6398-  60 60 00 40 31 00 00 00
63A0-  00 00 00 00 03 10 00 00
63A8-  00 00 00 00 00 1C 00 00
63B0-  3E 00 00 63 00 00 63 00
63B8-  00 63 00 00 63 00 00 63
63C0-  00 00 7F 00 00 7F 00 00
63C8-  73 00 00 73 00 00 7F 00
63D0-  00 00 00 00 00 00 03 10
63D8-  00 00 00 00 00 00 00 38
63E0-  00 00 7C 00 00 46 01 00
63E8-  46 01 00 46 01 00 46 01
63F0-  00 46 01 00 7E 01 00 7E
63F8-  01 00 46 01 00 46 01 00
6400-  7E 01 00 00 00 00 00 00
6408-  03 10 00 00 00 00 00 00
6410-  00 70 00 00 78 01 00 0C
6418-  03 00 06 06 00 06 06 00
6420-  06 06 00 06 06 00 7E 07
6428-  00 7E 07 00 06 06 00 06
6430-  06 00 0C 03 00 00 00 00
6438-  00 00 03 10 00 00 00 00
6440-  00 00 00 60 01 00 70 03
6448-  00 18 06 00 18 06 00 18
6450-  06 00 18 06 00 18 06 00
6458-  78 07 00 78 07 00 18 07
6460-  00 1C 0F 00 7E 1F 00 00
6468-  00 00 00 00
*
```

```
*6300.646B

6300-  00 0E 00 40 00 72 00 A4
6308-  00 D6 01 08 01 3A 03 10
6310-  00 00 00 00 00 00 54 0A
6318-  00 14 20 00 14 20 00 14
6320-  20 00 14 20 00 54 0A 00
6328-  54 0A 00 14 20 00 14 20
6330-  00 14 20 00 14 20 00 54
6338-  0A 00 00 00 00 00 00 00
6340-  03 10 00 00 00 00 00 00
6348-  28 15 00 28 40 00 28 40
6350-  00 28 50 00 28 50 00 28
6358-  04 00 28 04 00 28 44 00
6360-  28 44 00 28 40 00 28 40
6368-  00 28 15 00 00 00 00 00
6370-  00 00 03 10 00 00 00 00
6378-  00 00 50 2A 00 50 00 01
6380-  50 00 01 50 02 01 50 02
6388-  01 50 28 00 50 28 00 50
6390-  00 01 50 00 01 50 00 01
6398-  50 00 01 50 2A 00 00 00
63A0-  00 00 00 00 03 10 00 00
63A8-  00 00 00 00 20 55 00 20
63B0-  01 02 20 01 02 20 41 02
63B8-  20 41 02 20 11 00 20 11
63C0-  00 20 05 02 20 05 02 20
63C8-  01 02 20 01 02 20 55 00
63D0-  00 00 00 00 00 00 03 10
63D8-  00 00 00 00 00 00 40 2A
63E0-  01 40 02 04 40 02 04 40
63E8-  0A 04 40 0A 04 40 22 00
63F0-  40 22 00 40 02 05 40 02
63F8-  05 40 02 04 40 02 04 40
6400-  2A 01 00 00 00 00 00 00
6408-  03 10 00 00 00 00 00 00
6410-  00 55 02 00 05 08 00 05
6418-  08 00 05 08 00 05 08 00
6420-  45 02 00 45 02 00 15 08
6428-  00 15 08 00 05 08 00 05
6430-  08 00 55 02 00 00 00 00
6438-  00 00 03 10 00 00 00 00
6440-  00 00 00 2A 05 00 0A 10
6448-  00 0A 10 00 0A 11 00 0A
6450-  11 00 0A 01 00 0A 01 00
6458-  0A 11 00 0A 11 00 0A 10
6460-  00 0A 10 00 2A 05 00 00
6468-  00 00 00 00
*
```

```
*6300.64DB

6300- 00 0E 00 50 00 92 00 D4
6308- 01 16 01 58 01 9A 04 10
6310- 00 00 00 00 00 80 00 00
6318- A0 95 00 00 A8 D5 00 00
6320- A8 C0 80 00 88 00 80 00
6328- 88 00 80 00 88 00 80 00
6330- 88 00 80 00 88 00 80 00
6338- 88 00 80 00 A8 00 80 00
6340- A8 D5 80 00 A0 95 80 00
6348- 00 00 00 00 00 00 00 00
6350- 04 10 00 00 00 00 00 80
6358- 00 00 C0 AA 00 00 D0 AA
6360- 81 00 D0 AA 81 00 D0 AA
6368- 81 00 D0 AA 81 00 D0 AA
6370- 81 00 D0 82 80 00 D0 AA
6378- 81 00 D0 AA 81 00 D0 AA
6380- 81 00 D0 AA 81 00 C0 AA
6388- 80 00 00 00 00 00 00 00
6390- 00 00 04 10 00 00 00 00
6398- 00 80 00 00 80 D5 80 00
63A0- A0 D5 82 00 A0 D5 82 00
63A8- A0 81 80 00 A0 80 80 00
63B0- A0 80 80 00 A0 80 80 00
63B8- A0 80 80 00 A0 81 80 00
63C0- A0 95 80 00 A0 D5 82 00
63C8- 80 D5 80 00 00 00 00 00
63D0- 00 00 00 00 04 10 00 00
63D8- 00 00 00 00 00 00 00 AA
63E0- 81 00 C0 AA 85 00 C0 AA
63E8- 85 00 C0 AA 80 00 C0 8A
63F0- 80 00 C0 82 80 00 C0 82
63F8- 80 00 C0 8A 80 00 C0 AA
6400- 80 00 C0 AA 81 00 C0 AA
6408- 85 00 80 AA 81 00 00 00
6410- 00 00 00 00 00 00 04 10
6418- 00 00 00 00 00 00 80 00
6420- 80 D4 82 00 80 D5 8A 00
6428- 80 D5 8A 00 80 D5 80 00
6430- 80 95 80 00 80 85 80 00
6438- 80 85 80 00 80 95 80 00
6440- 80 D5 80 00 80 D5 82 00
6448- 80 D5 8A 00 80 D4 82 00
6450- 00 00 00 00 00 00 00 00
6458- 04 10 00 00 00 00 00 00
6460- 80 00 00 A8 85 00 00 AA
6468- 95 00 00 AA 95 00 00 8A
6470- 80 00 00 82 80 00 00 82
6478- 80 00 00 82 80 00 00 82
6480- 80 00 00 8A 80 00 00 AA
6488- 81 00 00 AA 95 00 00 A8
6490- 85 00 00 00 00 00 00 00
6498- 00 00 04 10 00 00 00 00
64A0- 00 00 80 00 00 D0 8A 00
64A8- 00 D4 AA 00 00 D4 AA 00
64B0- 00 D4 AA 00 00 D4 AA 80
64B8- 00 D4 AA 00 00 D4 80 80
64C0- 00 D4 AA 00 00 D4 AA 00
64C8- 00 D4 AA 00 00 D4 AA 00
64D0- 00 D0 8A 00 00 00 00 00
64D8- 00 00 00 00
*
```

```
*6300.646B

6300- 00 0E 00 40 00 72 00 A4
6308- 00 D6 01 08 01 3A 03 10
6310- 00 00 00 00 00 00 54 02
6318- 00 54 0A 00 04 08 00 04
6320- 28 00 04 20 00 04 20 00
6328- 04 20 00 04 20 00 04 28
6330- 00 04 08 00 54 0A 00 54
6338- 02 00 00 00 00 00 00 00
6340- 03 10 00 00 00 00 00 00
6348- 00 00 00 00 00 00 00 00
6350- 00 28 15 00 28 55 00 28
6358- 55 00 08 50 00 08 50 00
6360- 28 55 00 28 55 00 28 55
6368- 00 28 15 00 00 00 00 00
6370- 00 00 03 10 00 00 00 00
6378- 00 00 00 00 00 50 2A 00
6380- 50 2A 01 10 20 01 10 00
6388- 01 10 00 01 10 00 01 10
6390- 00 01 10 20 01 50 2A 01
6398- 50 2A 01 50 2A 00 00 00
63A0- 00 00 00 00 03 10 00 00
63A8- 00 00 00 00 00 00 00 00
63B0- 00 00 20 55 00 20 55 02
63B8- 20 55 02 20 40 02 20 00
63C0- 02 20 00 02 20 40 02 20
63C8- 55 02 20 55 02 20 55 00
63D0- 00 00 00 00 00 00 03 10
63D8- 00 00 00 00 00 00 00 00
63E0- 00 00 00 00 40 2A 01 40
63E8- 2A 05 40 2A 05 40 00 05
63F0- 40 00 04 40 00 04 40 00
63F8- 05 40 2A 05 40 2A 05 40
6400- 2A 01 00 00 00 00 00 00
6408- 03 10 00 00 00 00 00 00
6410- 00 00 00 00 55 02 00 55
6418- 0A 00 01 0A 00 01 08 00
6420- 01 08 00 01 08 00 01 08
6428- 00 01 0A 00 55 0A 00 55
6430- 0A 00 55 02 00 00 00 00
6438- 00 00 03 10 00 00 00 00
6440- 00 00 00 00 00 00 00 00
6448- 00 00 00 00 2A 05 00 2A
6450- 15 00 2A 15 00 02 14 00
6458- 02 14 00 2A 15 00 2A 15
6460- 00 2A 15 00 2A 05 00 00
6468- 00 00 00 00
*
]
```

```
*6300.646B

6300- 00 0E 00 40 00 72 00 A4
6308- 00 D6 01 08 01 3A 03 10
6310- 00 00 00 00 00 00 50 2A
6318- 00 54 2A 00 14 00 00 14
6320- 00 00 14 00 00 54 02 00
6328- 54 02 00 54 02 00 14 00
6330- 00 14 00 00 54 2A 00 50
6338- 2A 00 00 00 00 00 00 00
6340- 03 10 00 00 00 00 00 00
6348- 00 00 00 00 00 00 00 00
6350- 00 20 15 00 28 00 00 28
6358- 05 00 28 01 00 28 55 00
6360- 20 55 00 00 00 00 00 00
6368- 00 00 00 00 00 00 00 00
6370- 00 00 03 10 00 00 00 00
6378- 00 00 00 00 00 40 2A 01
6380- 50 2A 01 50 00 00 50 00
6388- 00 50 0A 00 50 0A 00 50
6390- 00 00 50 00 00 40 2A 00
6398- 00 00 00 00 00 00 00 00
63A0- 00 00 00 00 03 10 00 00
63A8- 00 00 00 00 00 00 00 00
63B0- 00 00 00 55 00 20 01 00
63B8- 20 01 00 20 15 00 20 15
63C0- 00 20 01 00 20 55 02 00
63C8- 55 02 00 00 00 00 00 00
63D0- 00 00 00 00 00 00 03 10
63D8- 00 00 00 00 00 00 00 00
63E0- 00 00 00 00 00 2A 05 40
63E8- 2A 05 40 02 00 40 2A 00
63F0- 40 2A 00 40 02 00 00 2A
63F8- 00 00 00 00 00 00 00 00
6400- 00 00 00 00 00 00 00 00
6408- 03 10 00 00 00 00 00 00
6410- 00 00 00 00 54 0A 00 55
6418- 0A 00 05 00 00 05 00 00
6420- 55 00 00 55 00 00 05 00
6428- 00 05 00 00 55 0A 00 54
6430- 0A 00 00 00 00 00 00 00
6438- 00 00 03 10 00 00 00 00
6440- 00 00 00 00 00 00 00 00
6448- 00 00 00 00 28 15 00 2A
6450- 15 00 2A 01 00 0A 00 00
6458- 28 01 00 00 00 00 00 00
6460- 00 00 00 00 00 00 00 00
6468- 00 00 00 00
*
```

```
*6300.646B

6300- 00 0E 00 40 00 72 00 A4
6308- 00 D6 01 08 01 3A 03 10
6310- 00 00 00 00 00 00 7C 3F
6318- 00 7C 3F 00 1C 00 00 1C
6320- 00 00 1C 00 00 1C 00 00
6328- 7C 0F 00 7C 0F 00 1C 00
6330- 00 1C 00 00 1C 00 00 1C
6338- 00 00 00 00 00 00 00 00
6340- 03 10 00 00 00 00 00 00
6348- 00 00 00 00 00 00 00 00
6350- 00 78 7F 00 78 7F 00 38
6358- 00 00 78 01 00 78 01 00
6360- 78 01 00 78 01 00 78 01
6368- 00 78 01 00 00 00 00 00
6370- 00 00 03 10 00 00 00 00
6378- 00 00 00 00 00 70 7F 01
6380- 70 7F 01 70 00 00 70 00
6388- 00 70 00 00 70 1F 00 70
6390- 1F 00 70 00 00 70 00 00
6398- 70 03 00 70 03 00 00 00
63A0- 00 00 00 00 03 10 00 00
63A8- 00 00 00 00 00 00 00 00
63B0- 00 00 60 7F 03 60 7F 03
63B8- 60 01 00 60 01 00 60 1F
63C0- 00 60 1F 00 60 01 00 60
63C8- 07 00 60 07 00 60 07 00
63D0- 00 00 00 00 00 00 03 10
63D8- 00 00 00 00 00 00 00 00
63E0- 00 00 00 00 40 7F 07 40
63E8- 7F 07 40 03 00 40 03 00
63F0- 40 3F 00 40 3F 00 40 03
63F8- 00 40 0F 00 40 0F 00 40
6400- 0F 00 00 00 00 00 00 00
6408- 03 10 00 00 00 00 00 00
6410- 00 00 00 00 7F 0F 00 7F
6418- 0F 00 07 00 00 07 00 00
6420- 07 00 00 7F 01 00 7F 01
6428- 00 07 00 00 07 00 00 1F
6430- 00 00 1F 00 00 00 00 00
6438- 00 00 03 10 00 00 00 00
6440- 00 00 00 00 00 00 00 00
6448- 00 00 00 00 7E 1F 00 7E
6450- 1F 00 0E 00 00 3E 00 00
6458- 3E 00 00 3E 00 00 3E 00
6460- 00 3E 00 00 3E 00 00 00
6468- 00 00 00 00
*
```

```
*6300.646B

6300-  00 0E 00 40 00 72 00 A4
6308-  00 D6 01 08 01 3A 03 10 ........
6310-  00 00 00 00 00 00 50 2A
6318-  00 54 2A 00 14 00 00 14
6320-  00 00 14 00 00 14 00 00
6328-  14 2A 00 14 2A 00 14 20
6330-  00 14 20 00 54 2A 00 50
6338-  2A 00 00 00 00 00 00 00
6340-  03 10 00 00 00 00 00 00 ........
6348-  20 55 00 28 55 00 28 00
6350-  00 28 00 00 28 00 00 28
6358-  04 00 28 14 00 28 14 00
6360-  28 50 00 28 50 00 28 55
6368-  00 20 15 00 00 00 00 00
6370-  00 00 03 10 00 00 00 00 ........
6378-  00 00 40 2A 01 50 2A 01
6380-  50 20 01 50 00 00 50 00
6388-  00 50 00 00 50 28 00 50
6390-  28 01 50 00 01 50 00 01
6398-  50 2A 01 40 2A 01 00 00
63A0-  00 00 00 00 03 10 00 00 ........
63A8-  00 00 00 00 00 55 02 20
63B0-  55 02 20 41 02 20 01 02
63B8-  20 01 00 20 01 00 20 51
63C0-  00 20 51 02 20 41 02 20
63C8-  41 02 20 55 02 00 55 02
63D0-  00 00 00 00 00 00 03 10 ........
63D8-  00 00 00 00 00 00 00 2A
63E0-  05 40 2A 05 40 02 05 40
63E8-  02 04 40 02 00 40 02 00
63F0-  40 02 01 40 22 05 40 22
63F8-  04 40 02 04 40 2A 05 00
6400-  2A 05 00 00 00 00 00 00
6408-  03 10 00 00 00 00 00 00 ........
6410-  00 54 0A 00 55 0A 00 05
6418-  0A 00 05 00 00 05 00 00
6420-  05 00 00 45 02 00 45 0A
6428-  00 05 08 00 05 08 00 55
6430-  0A 00 54 0A 00 00 00 00
6438-  00 00 03 10 00 00 00 00 ........
6440-  00 00 00 28 15 00 2A 15
6448-  00 0A 00 00 0A 00 00 0A
6450-  00 00 0A 00 00 0A 05 00
6458-  0A 15 00 0A 11 00 0A 11
6460-  00 2A 15 00 28 05 00 00
6468-  00 00 00 00
*
```

```
6300.646B

6300-  00 0E 00 40 00 72 00 A4
6308-  00 D6 01 08 01 3A 03 10 .......
6310-  00 00 00 00 00 00 14 28
6318-  00 14 28 00 14 28 00 14
6320-  28 00 14 28 00 54 2A 00
6328-  54 2A 00 14 28 00 14 28
6330-  00 14 28 00 14 28 00 14
6338-  28 00 00 00 00 00 00 00
6340-  03 10 00 00 00 00 00 00 .......
6348-  20 01 00 28 01 00 28 01
6350-  00 28 01 00 28 01 00 28
6358-  01 00 28 01 00 28 01 00
6360-  28 01 00 28 01 00 28 01
6368-  00 20 01 00 00 00 00 00
6370-  00 00 03 10 00 00 00 00 ......
6378-  00 00 50 02 00 50 22 00
6380-  50 22 00 50 22 00 50 22
6388-  00 50 2A 00 50 2A 00 50
6390-  22 00 50 22 00 50 22 00
6398-  50 22 00 50 02 00 00 00
63A0-  00 00 00 00 03 10 00 00 .....
63A8-  00 00 00 00 00 15 00 20
63B0-  15 00 20 15 00 20 15 00
63B8-  20 15 00 20 15 00 20 15
63C0-  00 20 15 00 20 15 00 20
63C8-  15 00 20 15 00 00 15 00
63D0-  00 00 00 00 00 00 03 10 ......
63D8-  00 00 00 00 00 00 40 0A
63E0-  00 40 2A 00 40 2A 00 40
63E8-  2A 00 40 2A 00 40 2A 00
63F0-  40 2A 00 40 2A 00 40 2A
63F8-  00 40 2A 00 40 2A 00 40
6400-  0A 00 00 00 00 00 00 00
6408-  03 10 00 00 00 00 00 00 ......
6410-  00 50 02 00 51 02 00 51
6418-  02 00 51 02 00 51 02 00
6420-  55 02 00 55 02 00 51 02
6428-  00 51 02 00 51 02 00 51
6430-  02 00 50 02 00 00 00 00
6438-  00 00 03 10 00 00 00 00 .......
6440-  00 00 00 0A 00 00 2A 00
6448-  00 2A 00 00 2A 00 00 2A
6450-  00 00 2A 00 00 2A 00 00
6458-  2A 00 00 2A 00 00 2A 00
6460-  00 2A 00 00 0A 00 00 00
6468-  00 00 00 00
*
```

```
6300.652F

6300-  00 0E 00 5C 00 AA 00 F8
6308-  01 46 01 94 01 E2 04 13 ........
6310-  00 00 00 00 00 00 00 00
6318-  40 03 00 00 40 03 00 00
6320-  40 03 00 00 40 03 00 00
6328-  40 03 00 00 40 03 00 00
6330-  40 03 00 00 40 03 00 00
6338-  40 03 00 00 40 03 00 00
6340-  40 03 00 00 40 03 00 00
6348-  40 03 00 00 00 00 00 00
6350-  00 00 00 00 00 00 00 00
```

```
6358-  00 00 00 00 04 13 00 00 .......
6360-  00 00 00 00 00 00 00 00
6368-  00 00 00 07 00 00 00 07
6370-  00 00 00 07 00 00 00 07
6378-  00 00 00 07 00 00 00 07
6380-  00 00 00 07 00 00 00 07
6388-  00 00 00 0F 00 00 00 1F
6390-  00 00 00 1E 00 00 00 1C
6398-  00 00 00 00 00 00 00 00
63A0-  00 00 00 00 00 00 00 00
```

```
63A8-  00 00 04 13 00 00 00 00 .......
63B0-  00 00 00 00 00 00 00 00
63B8-  00 0E 00 00 00 0E 00 00
63C0-  00 0E 00 00 00 0E 00 00
63C8-  00 0E 00 00 00 0E 00 00
63D0-  00 0F 00 00 00 0F 00 00
63D8-  00 0F 00 00 00 0F 00 00
63E0-  40 07 00 00 60 07 00 00
63E8-  00 00 00 00 00 00 00 00
63F0-  00 00 00 00 00 00 00 00
```

```
63F8-  04 13 00 00 00 00 00 00 .......
6400-  00 00 00 00 00 00 00 00
6408-  00 00 00 1C 00 00 00 1C
6410-  00 00 00 1C 00 00 00 1C
6418-  00 00 00 1C 00 00 00 1C
6420-  00 00 00 3C 00 00 00 78
6428-  00 00 00 78 01 00 00 70
6430-  03 00 00 60 07 00 00 00
6438-  00 00 00 00 00 00 00 00
```

```
6440-  00 00 00 00 00 00 04 13 ........
6448-  00 00 00 00 00 00 00 00
6450-  00 00 00 00 00 00 00 00
6458-  00 38 00 00 00 38 00 00
6460-  00 38 00 00 00 38 00 00
6468-  00 38 00 00 00 38 00 00
6470-  00 3C 00 00 00 1E 00 00
6478-  00 1F 00 00 00 0F 00 00
6480-  40 07 00 00 00 00 00 00
6488-  00 00 00 00 00 00 00 00
```

```
6490-  00 00 00 00 04 13 00 00 .........
6498-  00 00 00 00 00 00 00 00
64A0-  00 00 00 70 00 00 00 70
64A8-  00 00 00 70 00 00 00 70
64B0-  00 00 00 70 00 00 00 70
64B8-  00 00 00 70 00 00 00 70
64C0-  00 00 00 70 01 00 00 70
64C8-  03 00 00 60 03 00 00 40
64D0-  03 00 00 00 00 00 00 00
64D8-  00 00 00 00 00 00 00 00
```

```
64E0-  00 00 04 13 00 00 00 00 .........
64E8-  00 00 00 00 00 00 00 00
64F0-  00 60 01 00 00 60 01 00
64F8-  00 60 01 00 00 60 01 00
6500-  00 60 01 00 00 60 01 00
6508-  00 70 01 00 00 70 01 00
6510-  00 70 01 00 00 70 01 00
6518-  00 78 00 00 00 7C 00 00
6520-  00 00 00 00 00 00 00 00
6528-  00 00 00 00 00 00 00 00
*
```

```
6300.646B

6300-  00 0E 00 40 00 72 00 A4
6308-  00 D6 01 08 01 3A 03 10
6310-  00 00 00 00 00 00 D4 AA
6318-  00 D4 AA 00 80 8A 00 80
6320-  8A 00 80 8A 00 80 8A 00
6328-  80 8A 00 80 8A 00 84 9A
6330-  00 84 8A 00 D4 8A 00 D4
6338-  8A 00 00 00 00 00 00 00
6340-  03 10 00 00 00 00 00 00
6348-  A6 D5 00 A8 D5 00 00 85
6350-  00 00 85 00 00 85 00 00
6358-  85 00 00 85 00 00 85 00
6360-  00 C5 00 00 C5 00 80 D5
6368-  00 00 D5 00 00 00 00 00
6370-  00 00 03 10 00 00 00 00
6378-  00 00 D0 AA 81 80 AA 81
6380-  00 8A 00 00 8A 00 00 8A
6388-  00 00 8A 00 00 8A 00 00
6390-  8A 00 90 8A 00 90 8A 00
6398-  D0 8A 00 D0 8A 00 00 00
63A0-  00 00 00 00 03 10 00 00
63A8-  00 00 00 00 A0 D5 82 A0
63B0-  D5 82 00 94 00 00 94 00
63B8-  00 94 00 00 94 00 00 94
63C0-  00 00 94 00 00 94 00 00
63C8-  94 00 00 94 00 00 94 00
63D0-  00 00 00 00 00 00 03 10
63D8-  00 00 00 00 00 00 C0 AA
63E0-  85 C0 AA 85 00 A8 00 00
63E8-  A8 00 00 A8 00 00 A8 00
63F0-  00 A8 00 00 A8 00 00 A8
63F8-  00 00 A8 00 00 A8 80 00
6400-  A8 80 00 00 00 00 00 00
6408-  03 10 00 00 00 00 00 00
6410-  00 D5 8A 00 D5 8A 00 D0
6418-  00 00 D0 00 00 D0 00 00
6420-  D0 00 00 D0 00 00 D0 00
6428-  00 D1 00 00 D1 00 00 D5
6430-  00 00 D5 00 00 00 00 00
6438-  00 00 03 10 00 00 00 00
6440-  00 00 00 AA 95 00 AA 95
6448-  00 A0 81 00 A0 81 00 A0
6450-  81 00 A0 81 00 A0 81 00
6458-  A0 81 00 A0 91 00 A0 91
6460-  00 A0 95 00 A0 95 00 00
6468-  00 00 00 00
*
```

```
6300.64D8

6300-  00 0E 00 50 00 92 00 D4
6308-  01 16 01 58 01 9A 04 10
6310-  00 00 00 00 00 00 00 00
6318-  28 40 00 00 28 50 00 00
6320-  28 54 00 00 28 14 00 00
6328-  28 05 00 00 28 05 00 00
6330-  28 05 00 00 28 15 00 00
6338-  28 15 00 00 28 54 00 00
6340-  28 50 00 00 28 50 00 00
6348-  00 00 00 00 00 00 00 00
6350-  04 10 00 00 00 00 00 00
6358-  00 00 50 00 00 00 50 00
6360-  00 00 50 00 00 00 50 00
6368-  00 00 50 0A 00 00 50 2A
6370-  01 00 50 2A 01 00 50 02
6378-  00 00 50 00 00 00 50 00
6380-  00 00 50 00 00 00 50 00
6388-  00 00 00 00 00 00 00 00
6390-  00 00 04 10 00 00 00 00
6398-  00 00 00 00 20 01 00 00
63A0-  20 01 00 00 20 41 02 00
63A8-  20 51 02 00 20 55 00 00
63B0-  20 05 00 00 20 05 00 00
63B8-  20 15 00 00 20 11 00 00
63C0-  20 51 00 00 20 51 02 00
63C8-  20 01 00 00 00 00 00 00
63D0-  00 00 00 00 04 10 00 00
63D8-  00 00 00 00 00 00 40 02
63E0-  00 00 40 02 00 00 40 02
63E8-  00 00 40 22 05 00 40 2A
63F0-  05 00 40 2A 05 00 40 0A
63F8-  00 00 40 2A 00 00 40 22
6400-  00 00 40 02 01 00 40 02
6408-  00 00 40 02 00 00 00 00
6410-  00 00 00 00 00 00 04 10
6418-  00 00 00 00 00 00 02 00
6420-  00 05 00 00 00 05 [illegible] 00
6428-  00 05 0A 00 00 45 02 00
6430-  00 55 00 00 00 55 00 00
6438-  00 55 0A 00 00 55 0A 00
6440-  00 45 0A 00 00 45 02 00
6448-  00 05 00 00 00 05 [illegible] 00
6450-  00 00 00 00 00 00 [illegible] 00
6458-  04 10 00 00 00 00 [illegible] 00
6460-  00 00 00 0A 00 00 00 0A
6468-  00 00 00 0A 14 00 00 0A
6470-  15 00 00 2A 01 00 00 2A
6478-  00 00 00 2A 00 00 00 2A
6480-  01 00 00 0A 01 00 00 0A
6488-  05 00 00 0A 15 00 00 0A
6490-  00 00 00 00 03 00 00 00
6498-  00 00 04 10 00 00 00 00
64A0-  00 00 00 00 00 14 00 00
64A8-  00 14 00 00 00 14 00 00
64B0-  00 14 0A 00 00 54 00 00
64B8-  00 54 2A 00 00 54 2A 00
64C0-  00 54 2A 00 00 14 2A 00
64C8-  00 14 00 00 00 14 00 00
64D0-  00 14 00 00 00 00 00 00
64D8-  00 00 00 00
*
```

```
6300.646B

6300-  00 0E 00 40 00 72 00 A4
6308-  00 D6 01 08 01 3A 03 10
6310-  00 00 00 80 00 00 A8 00
6318-  00 A8 00 00 A8 00 00 A8
6320-  00 00 A8 00 00 A8 00 00
6328-  A8 00 00 A8 00 00 A8 00
6330-  00 A8 00 00 A8 95 00 A8
6338-  95 00 80 00 00 00 00 00
6340-  03 10 00 00 00 00 00 00
6348-  D0 00 00 D0 00 00 D0 82
6350-  00 D0 82 00 C0 82 00 C0
6358-  82 00 C0 AA 00 00 A8 00
6360-  00 A8 00 00 A0 00 00 A0
6368-  81 00 A0 81 00 00 00 00
6370-  00 00 03 10 00 00 00 A0
6378-  81 00 A0 81 00 A0 81 00
6380-  A0 81 00 A0 81 00 A0 81
6388-  00 A0 81 00 A0 81 00 A0
6390-  81 00 A0 81 00 80 85 00
6398-  00 D5 80 00 D0 80 00 00
63A0-  00 00 00 00 03 10 00 00
63A8-  00 00 00 00 C0 82 00 C0
63B0-  82 00 C0 82 00 C0 82 00
63B8-  C0 82 00 C0 8A 00 00 8A
63C0-  00 00 8A 00 00 AA 00 00
63C8-  A8 00 00 A8 81 00 A8 81
63D0-  00 00 00 00 00 00 03 10
63D8-  00 00 00 00 00 00 80 85
63E0-  00 80 85 00 80 85 00 80
63E8-  85 00 00 85 00 00 95 00
63F0-  00 94 00 00 D4 00 00 D4
63F8-  80 00 D0 80 00 D0 82 00
6400-  D0 82 00 00 00 00 00 00
6408-  03 10 00 00 00 00 00 00
6410-  00 8A 00 00 8A 00 00 8A
6418-  00 00 8A 00 00 8A 00 00
6420-  8A 00 00 8A 00 00 88 00
6428-  00 A8 81 00 A8 81 00 A0
6430-  85 00 80 85 00 00 00 00
6438-  00 00 03 10 00 00 00 00
6440-  00 00 00 34 00 00 94 00
6448-  00 D4 80 C0 D0 80 00 D0
6450-  80 00 D0 80 00 D0 8A 00
6458-  00 8A 00 00 8A 00 00 88
6460-  00 00 A8 00 00 A8 00 00
6468-  00 00 00 00
*
```

```
6300.646B

6300-  00 0E 00 40 00 72 00 A4
6308-  00 DC 01 08 01 3A 03 10
6310-  00 00 00 00 00 00 1C 30
6318-  00 3C 3C 00 7C 3F 00 7C
6320-  3F 00 7C 3F 00 7C 3F 00
6328-  4C 33 00 4C 33 00 4C 33
6330-  00 4C 33 00 0C 30 00 0C
6338-  30 00 00 00 00 00 00 00
6340-  03 10 00 00 00 00 00 00
6348-  40 0F 00 40 0F 00 40 0F
6350-  00 40 0F 00 40 0F 00 40
6358-  0F 00 40 0F 00 78 7F 00
6360-  78 7F 00 78 7F 00 78 7F
6368-  00 78 7F 00 00 00 00 00
6370-  00 00 03 10 00 00 00 00
6378-  00 00 60 60 00 60 71 00
6380-  60 7F 00 60 7F 00 60 7F
6388-  00 60 7F 00 60 6E 00 60
6390-  6E 00 60 6E 00 60 6E 00
6398-  60 60 00 70 60 01 00 00
63A0-  00 00 00 00 03 10 00 00
63A8-  00 00 00 00 00 41 00 00
63B0-  63 00 00 7F 00 00 7F 00
63B8-  00 7F 00 00 7F 00 00 7F
63C0-  00 00 7F 00 00 7F 00 60
63C8-  7F 03 60 77 03 60 63 03
63D0-  00 00 00 00 00 00 03 10
63D8-  00 00 00 00 00 00 00 02
63E0-  01 00 46 01 00 7E 01 00
63E8-  7E 01 00 7E 01 00 7E 01
63F0-  00 7E 01 00 7E 01 00 7E
63F8-  01 40 7F 07 40 6F 07 40
6400-  47 07 00 00 00 00 00 00
6408-  03 10 00 00 00 00 00 00
6410-  00 06 06 00 0E 07 00 7E
6418-  07 00 7E 07 00 7E 07 00
6420-  7E 07 00 76 06 00 76 06
6428-  00 76 06 00 76 06 00 06
6430-  06 00 07 0E 00 00 00 00
6438-  00 00 03 10 00 00 00 00
6440-  00 00 00 70 03 00 70 03
6448-  00 70 03 00 70 03 00 70
6450-  03 00 70 03 00 70 03 00
6458-  7E 1F 00 7E 1F 00 7E 1F
6460-  00 7E 1F 00 5E 1F 00 00
6468-  00 00 00 00
*
```

F 計分程式

除了畫圖之外，通常我們也需要在畫面上添加上一些文字，當然我們可以將一些造字副程式及字形表附加到程式來便可以了。可是畢竟動畫程式是以圖面爲主，文字的使用並不多，甚至不必將整組的字形表均列入程式裡頭。一些 Game的計分方式便是：

圖 F−0

在圖 F − 0 當中，像HI SCORE這類的文字只使用了少數的英文字母，並且是在程式一開始便已畫好了。而程式執行當中，只有計分用的數字是需要隨時變動的，所以這樣的計分程式，其重點便只有在於如何將機械語言的16進位數變換成10進位數而印出到螢幕上如此而已。底下便是一個計數及進位的副程式：

```
;
*****************
* SCORE ROUTINE *
*****************
*
            ORG $1000
```

```
*
SCORE     INC KILLNUM   ；打下一個外星人
          INC SCOREA    ；計數加一
          LDA SCOREA
          CMP #$0A      ；是否個位數大於 10 ？
          BLT SCRSET    ；尚未，不進位
          LDA #$00      ；進位，個位數歸 0
          STA SCOREA
SCORE10   INC SCOREB    ；十位數加一
          LDA SCOREB
          CMP #$0A      ；十位數大於 10 ？
          BLT SCRSET    ；< 10 ，不進位
          LDA #$00      ；進位， 十位數歸 0
          STA SCOREB
          INC SCOREC    ；百位數加一
          LDA SCOREC
          CMP #$0A      ；百位數大於 10 ？
          BLT SCRSET
          LDA #$00      ；若大於 999 ？
          STA SCOREA    ；則由 0 開始再
          STA SCOREB
          STA SCOREC
*
* SCORE SETUP ROUTINE FOR DRAW
*
```

程式 F - 0　十進位的計數程式

程式 F － 0 以 SCOREA 、 SCOREB 、 SCOREC 分別來代表分數的個、十、百位數值，請參考圖 F － 0 的說明。由於分數所顯示的位置固定不變，因此只要將新的數值疊到舊的數字上面便可以更新分數的記錄了。

當然我們必須造一個字形表，以便印字時取用，你可以設計自己喜歡的字形（ 1 、 I 、 i 、一、壹……？）。當走完程式 F － 0 的副程式後，必須呼叫 DRAW 將 SCOREA 、 SCOREB 、 SCOREC 的內含值印出才行，這便是計分程式了。

參考文獻

1. Graph paper, David Lubar, Creative Computing Feb & March 1983
2. Enhancing Your Apple Ⅱ（波前）1983
3. Apple Graphic & Arcade Game Design, Jeffrey stanton 1982
4. Fudge it, Don Fudge, In Cider, May 1983
5. The Complete Graphic System, Pelczarski 1982
6. The paddle program, R.A.Cox, Apple Orchard Feb 1983

著作權、版權所有翻印必究

法律顧問：邱晃泉律師

APPLE Ⅱ微電腦徹底研究(五)

作　者：施威銘
地　址：台北縣板橋市林森街一巷十一號
電　話：9570111
出版者：波前電腦管理圖書有限公司
發行人：劉麗雪
地　址：台北市復興北路 179 號 3 樓
電　話：7122248-9
郵　撥：561411　　　　帳戶：劉麗雪

定　價：國內：台幣 220 元
　　　　國外：港幣 42 元

印刷者：名揚彩色印刷有限公司
著作權字號：
中華民國 72 年 8 月再版

介紹您：波前的新書

APPLE II 繪圖遊戲系列：

- 從APPLE II 到繪圖遊戲製作
  宗世麟 著
- APPLE II 繪圖遊戲系統設計
  林和賢 著

眞正的繪圖遊戲，
眞正的Graphic Game

APPLE II 硬體系列：

- APPLE II 硬體線路分析
  施威銘 校訂 吳占鰲•李士彬 編譯
- APPLE II 自動控制實務設計
  吳占鰲 著

徹底的硬體分析，
滿足最挑剔的玩家要求

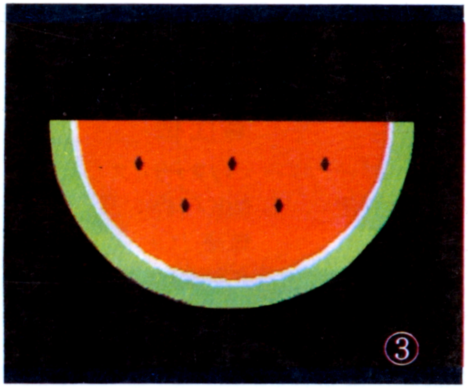
• 波前出版 • 必屬好書 •

這本書……

- 告訴您Apple的顏色有多少？
- 告訴您最新的職業動畫技巧。
- 告訴您如何結構化的來寫動畫程式。
- 從此，機械語言的Game，對您來說再也不是困難的事情了
- 從此，您可以自行設計專業化的遊戲或教學程式了！

①

②



③

波前電腦管理圖書有限公司

APPLE II

微電腦徹底研究 5

動畫技巧篇

施威銘 著

波前